WROCŁAW UNIVERSITY OF SCIENCE AND TECHNOLOGY

Non-stationary data stream processing with metafeature analysis

by

Joanna Komorniczak

A dissertation submitted in partial fulfillment
for the degree of Doctor of Philosophy

in the
Faculty of Information and Communication Technology
Department of Systems and Computer Networks

April 2025

Streszczenie

Rozprawa doktorska koncentruje się na zagadnieniach związanych z przetwarzaniem niestacjonarnych strumieni danych. Proponowane w rozprawie metody uczenia maszynowego pozwalają na rozwiazywanie dwóch kluczowych dla tej dziedziny zadań: *detekcji dryfów koncepcji* oraz *klasyfikacji niestacjonarnych strumieni danych* w oparciu o *metody hybrydowe*. Zaproponowane oraz zbadane w rozprawie rozwiązania opierały się na analizie *metacech* obliczanych dla rozłącznych wsadów napływających w formie strumieni danych. Celem rozprawy była weryfikacja hipotezy:

> *Możliwa jest propozycja metod wykorzystujących analizę metacech na potrzeby detekcji dryfów koncepcji oraz klasyfikacji niestacjonarnych strumieni danych, które wykazują jakość rozpoznawania istotnie lepszą lub statystycznie zależną od podejść znanych z literatury.*

W oparciu o przeprowadzone badania hipoteza została uprawdopodobniona poprzez osiągnięcie następujących celów:

Cel 1: Propozycja niejawnego detektora dryfu koncepcji analizującego zmienność w czasie miar złożoności zadania klasyfikacji obliczonych dla rozłącznych wsadów danych. Cel został zrealizowany poprzez propozycję metody *Complexity-based Drift Detector*, która analizuje metacechy opisujące złożoność zadania klasyfikacji. Działanie mechanizmu detekcji jest niezależne od wyników osiąganych przez model klasyfikacji, co pozwala na niejawną detekcję dryfu koncepcji. Zaproponowane podejście wykorzystuje zespół klasyfikatorów jednoklasowych szkolonych za pomocą metacech w celu rozróżnienia reprezentacji wsadów pochodzących z różnych koncepcji, co pozwala na efektywne rozpoznawanie istotnych zmian zachodzących w strumieniach danych.

Cel 2: Propozycja niejawnego detektora dryfu koncepcji analizującego miary magnitudy dryfu zintegrowane zgodnie z paradygmatem uczenia zespołowego. Cel został zrealizowany poprzez propozycję metody *Statistical Drift Detection Ensemble*, wykorzystującej miary *drift magnitude* oraz *conditioned marginal covariate drift* obliczane dla rozłącznych wsadów danych. Metoda detekcji dryfu wykorzystuje podejście zespołowe do integracji składowych decyzji o rozpoznaniu zmiany koncepcji.

Metacechy wykorzystane w tej metodzie dedykowane są analizie zmian koncepcji w strumieniach danych, jawnie opisując zmiany w rozkładzie prawdopodobieństwa. Bezpośrednie wykorzystanie tych miar wykazało ich niską informatywność w przypadku analizy danych wielowymiarowych. Zaproponowane w detektorze podejście zespołowe pozwala na efektywne rozpoznawanie dryfów dzięki analizie podprzestrzeni o niskiej wymiarowości i następnie integrację składowych detekcji w obrębie podprzestrzeni wykorzystując paradygmat uczenia zespołowego.

Cel 3: Propozycja nienadzorowanego detektora dryfu koncepcji analizującego rozkład aktywacji pochodzących z ostatniej warstwy deterministycznej sieci neuronowej. Cel został zrealizowany poprzez propozycję metody *Parallel Activations Drift Detector* pozwalającej na nienadzorowaną detekcję dryfów koncepcji w oparciu o wyjścia z deterministycznej sieci neuronowej. Wykorzystane w zaproponowanym podejściu metacechy zdefiniowane są jako losowe projekcje realizowane przez sieć neuronową zainicjalizowaną losowymi wagami, niezmiennymi w trakcie przetwarzania. Metacechy określone w ten sposób niejawnie opisują położenie próbek w wielowymiarowej przestrzeni. Analiza ich zmienności za pomocą wielokrotnie powtórzonych parowych testów statystycznych pozwala na identyfikację istotnych zmian w rozkładzie danych wskazujących na dryfy koncepcji.

Ze względu na brak wiarygodnego kryterium oceny jakości zadania detekcji dryfów koncepcji w literaturze, które pozwoliłoby na zbadanie jakości rozwiązania w przypadku strumieni danych z dryfami inkrementalnymi i gradualnymi, w rozprawie doktorskiej zostały zaproponowane trzy miary oceny jakości *błędu detekcji dryfu* bazujące na odległości oraz liczności dryfów i detekcji. Dodatkowo ocena jakości wspomagana była wizualną analizą momentów sygnalizacji zmian koncepcji.

Porównanie wszystkich zaproponowanych metod detekcji dryfów koncepcji z innymi detektorami znanymi z literatury na szerokiej puli syntetycznych strumieni danych pozwoliło na wykazanie, w zależności od badanego kryterium, istotnie lepszej lub statystycznie zależnej jakości detekcji w porównaniu do metod znanych z literatury.

Cel 4: Propozycja metody zespołowej do klasyfikacji strumieni danych analizującej rozkład statystycznych metacech obliczonych dla kolejnych wsadów danych w celu identyfikacji nawracającego konceptu. Cel został zrealizowany poprzez propozycję metody *Metafeature Concept Selector*, która wykorzystuje zestaw statystycznych metacech do detekcji dryfów koncepcji oraz ponownej identyfikacji koncepcji występującej w przeszłości, w przypadku koncepcji nawracających. Zaproponowany algorytm wykorzystuje pulę klasyfikatorów jednoklasowych do identyfikacji koncepcji na podstawie obliczonych metacech oraz, równolegle, pulę klasyfikatorów odpowiedzialnych za efektywne rozwiązywanie realizowanego zadania rozpoznawania.

W eksperymentach badających działanie metody wykorzystano ocenę jakości identyfikacji koncepcji nawracającej za pomocą metryki *Rand* dedykowanej ocenie zadań klasteryzacji. Metryka pozwoliła na jednoznaczną ocenę realizowanego zadania niezależnie od mechanizmu klasyfikacji zintegrowanego klasyfikatora. W dalszych eksperymentach porównano działanie zaproponowanego rozwiązania z działaniem niezależnych klasyfikatorów, co wykazało statystycznie znaczącą poprawę jakości klasyfikacji przy użyciu zaproponowanego mechanizmu.

Cel 5: Propozycja metody klasyfikacji pozwalającej na kompensację stronniczości klasyfikatorów podczas przetwarzania strumieni danych z dynamicznymi zmianami stopnia niezbalansowania wykorzystujących estymację prawdopodobieństwa a priori w oparciu o metacechy przetwarzanych porcji danych. Cel został zrealizowany poprzez propozycję *Prior Probability Assisted Classifier*, wykorzystujący estymowane prawdopodobieństwo *a priori* w celu kompensacji stronniczości klasyfikatorów w kierunku klasy większościowej. Estymacja prawdopodobieństwa a priori odbywa się w oparciu o metacechy obliczane dla kolejnych wsadów strumieni danych. W oparciu o estymowany stopień niezbalansowania w danym wsadzie, zaproponowana metoda koryguje predykcje klasyfikatora zapewniając określoną liczność obiektów danej klasy.

W eksperymentach porównano różne strategie estymacji prawdopodobieństwa *a priori*, z których najbardziej uniwersalną stanowiła *Dynamic Statistical Concept Analysis*, będąca autorską metodą określającą aktualne prawdopodobieństwo *a priori* problemu w oparciu o parę regresorów analizujących wartości średnie i odchylenie standardowe próbek w obrębie przetwarzanych klas. Wyniki badań wykazały statystycznie znaczącą poprawę jakości klasyfikacji przy zastosowaniu zaproponowanego podejścia.

Cel 6: Propozycja struktury przetwarzania strumieni danych ze zmiennym w czasie stopniem trudności, pozwalającej na wybór odpowiedniej architektury sieci neuronowej na postawie analizy pewności modelu. Cel został zrealizowany poprzez propozycję *Certainty-based Architecture Selection Framework* – struktury przetwarzania wykorzystującej stopień *pewności* modelu klasyfikacji w zadaniu klasyfikacji strumieni danych o zmiennej w czasie trudności. Wykorzystanie *pewności* klasyfikatora jako metacechy pozwoliło na redukcję narzutu obliczeniowego związanego z określeniem dodatkowych wskaźników opisujących przetwarzane dane. Na podstawie wartości *funkcji wsparcia* pochodzących z sieci neuronowej obliczane były metacechy, a następnie predykcje dla przetwarzanych obiektów.

W przeprowadzonych eksperymentach realizowane było zadanie klasyfikacji półsyntetycznych strumieni danych wizji komputerowej. Zaproponowany schemat przetwarzania pozwala na przełączanie architektur *konwolucyjnych sieci neuronowych* o różnym stopniu złożoności, wyszkolonych dla zadanego problemu. Wyniki przeprowadzonych eksperymentów pokazały, że zaproponowane rozwiązanie pozwala na znaczące zredukowanie czasu przetwarzania oraz liczby operacji przy niewielkiej redukcji jakości rozpoznawania.

Słowa kluczowe uczenie maszyn, strumienie danych, dryf koncepcji, detekcja dryfu koncepcji, klasyfikacja, zespoły klasyfikatorów, metacechy, dane niezbalansowane

Abstract

The doctoral dissertation focuses on topics related to non-stationary data stream processing. The machine learning methods proposed in the dissertation allow for solving two key tasks in the considered field: *concept drift detection* and *classification of non-stationary data streams* using *hybrid* approaches. The solutions proposed and examined in the dissertation are based on the analysis of *metafeatures* calculated for disjoint data batches arriving in the form of a stream. The dissertation aimed to verify the following hypothesis:

> *It is possible to propose methods employing metafeature analysis for concept drift detection and classification of the non-stationary data streams that demonstrate significantly better or statistically dependent recognition quality compared to state-of-the-art approaches.*

Based on the conducted research, the hypothesis was substantiated by achieving the following objectives:

Objective 1: Proposal of an implicit concept drift detector analyzing the time variability of the classification task complexity measures calculated for disjoint data chunks. The objective was achieved by proposing the *Complexity-based Drift Detector*, which analyzes metafeatures describing the complexity of the classification task. The detection mechanism operates independently of the classification quality, allowing for *implicit* concept drift detection. The proposed approach uses an ensemble of one-class classifiers trained with data metafeatures to distinguish between representations of batches from different concepts, allowing for effective recognition of changes occurring in the data streams.

Objective 2: Proposal of an implicit concept drift detector analyzing drift magnitude measures integrated using the ensemble learning paradigm. The objective was achieved by proposing the *Statistical Drift Detection Ensemble*, which uses *drift magnitude* and *conditioned marginal covariate drift* measures calculated for disjoint data batches of a data stream. The drift detection method uses an ensemble approach to integrate constituent decisions of the concept change recognition.

The metafeatures used in this method are dedicated to analyzing concept changes in data streams, directly indicating probability distribution shifts. The original measures showed low informativeness in the case of processing high-dimensional data. The ensemble approach proposed in the detector allows for effective drift recognition by analyzing low-dimensional subspaces and the integration of constituent detections within the subspaces using the ensemble learning paradigm.

Objective 3: Proposal of an unsupervised drift detection method analyzing the distribution of activations from the last layer of a deterministic neural network. The objective was achieved by proposing the *Parallel Activations Drift Detector* method, which allows for unsupervised detection of concept drifts based on the outputs of a deterministic neural network. The metafeatures used in the proposed approach are defined as random projections generated using the neural network initialized with random weights, invariant during the processing. The metafeatures defined in such a way implicitly describe the location of samples in multidimensional feature space. Analysis of their variability using replicated paired statistical tests allows for identifying significant changes in the data distribution that indicate concept drifts.

Due to the lack of a reliable criterion for assessing the quality of the concept drift detection task in the literature, which would allow for examining the quality of methods in the case of data streams with incremental and gradual drifts, three *drift detection error* measures were proposed in the dissertation. The proposed evaluation criteria employ the analysis of distance and the cardinality of drifts and detections. Additionally, the quality assessment was supported by a visual analysis of the moments of concept change signaling.

Comparison of all proposed concept drift detection methods with other *state-of-the-art* approaches on an extensive pool of synthetic data streams allowed for demonstrating, depending on the examined criterion, significantly better or statistically dependent detection quality compared to the methods known from the literature.

Objective 4: Proposal of an ensemble method for classification of data streams analyzing the distributions of statistical metafeatures calculated for subsequent data chunks to identify recurring concepts. The objective was achieved by proposing the *Metafeature Concept Selector* method, which uses a set of statistical metafeatures to detect concept drifts and to re-identify concepts that occurred in the past in the case of their recurrence. The proposed algorithm uses a pool of one-class classifiers to distinguish concepts based on the calculated metafeatures and, in parallel, a pool of classifiers responsible for performing the recognition task.

In the experiments examining the method's operation, the quality of concept identification was assessed using the *Rand* metric dedicated to the evaluation of clustering tasks. The metric allowed for an unambiguous and reliable assessment of the concept recognition quality, regardless of the classification mechanism of the baseline classifier. In further experiments, the performance of the proposed solution was compared with independent classifiers, which showed a statistically significant improvement in the classification quality when using the proposed approach.

Objective 5: Proposal of a classification method for compensating the bias of baseline classifiers when processing the data streams with dynamic changes in imbalance ratio, using the prior probability estimated based on the metafeatures of the processed data chunk. The objective was achieved by proposing *Prior Probability Assisted Classifier*, which uses an estimated prior probability to compensate for the classifier's bias towards the majority class. The estimation of prior probability is based on metafeatures calculated for subsequent batches of data streams. Based on the estimated level of imbalance in a given data batch, the proposed method corrects the predictions of the baseline classifier, ensuring a specific number of objects of a given class.

In the experiments, different strategies for estimating the prior probability were compared. The most universal approach was the *Dynamic Statistical Concept Analysis*, an original method estimating the current prior probability based on a pair of regressors analyzing the mean values and the standard deviation of samples within the processed classes. The results showed a statistically significant improvement in the classification quality when using the proposed method.

Objective 6: Proposal of a framework for processing data streams with a time-varying level of difficulty, allowing for the selection of an appropriate neural network architecture based on the analysis of the model's certainty. The objective was achieved by proposing a *Certainty-based Architecture Selection Framework* – a processing scheme that uses the classification model's *certainty* level in the classification task of data streams characterized by time-varying difficulty. Using the *certainty* of the classifier as a metafeature allows for avoiding the computational overhead associated with defining additional indicators describing the data. The *support function* values provided by the neural network were used to calculate the metafeatures and then to determine the predictions for the processed objects.

The performed experiments focused on the classification of semi-synthetic computer vision data streams. The proposed processing scheme showed the ability to dynamically switch architectures of *convolutional neural networks* of different complexities, trained for the given problem. The results of the experiments showed that the proposed solution allows for a significant reduction in processing time complexity and the number of operations of the system with a subtle reduction in recognition quality.

Keywords machine learning, data streams, concept drift, concept drift detection, classification, ensemble classifiers, metafeatures, imbalanced data

Abbreviations

Methods

GNB	**G**aussian **N**aive **B**ayes Classifier
KNN	**k**-**N**earest **N**eighbors Classifier
DT	**D**ecision **T**ree Classifier
MLP	**M**ultilayer **P**erceptron
SVM	**S**upport **V**ector **M**achine
C2D	**C**omplexity-based **D**rift **D**etector
SDDE	**S**tatistical **D**rift **D**etection **E**nsemble
PADD	**P**arallel **A**ctivations **D**rift **D**etector
MCS	**M**etafeature **C**oncept **S**elector
2PAC	**P**rior **P**robability **A**ssisted **C**lassifier
CAS	**C**ertainty-based **A**rchitecture **S**election

Metrics

ACC	**Acc**uracy
BAC	**B**alanced **Ac**curacy
RI	**R**and **I**ndex
DM	**D**rift **M**agnitude
CMCD	**C**onditioned **M**arginal **C**ovariate **D**rift
PD	**P**osterior **D**rift

D1	Average **D**istance from each Detection to the Nearest Drift
D2	Average **D**istance of each Drift to the Nearest Detection
R	Adjusted **R**atio of the Number of Drifts to the Number of Detections
TTAL	**T**ime **t**o **A**ccuracy **L**oss

Symbols

x	sample
d	dimensionality
x_k	k^{th} sample
x^k	k^{th} feature
$\mathcal{X}$	feature space
$\mathcal{Y}$	set of labels
M	number of labels
y_k	k^{th} class label
$\mathcal{D}$	dataset
n	number of samples
Ψ	classification function
X_k	decision area of k^{th} class
Π	ensemble of classifiers
f	metafeature
p	dimensionality of a metafeature
m	characterization measure
p'	dimensionality of a characterization measure
σ	summarization function

$\mathcal{DS}$	data stream
$\mathcal{DS}_k$	k^{th} data chunk
$P(\mathcal{X},\mathcal{Y})$	joint probability
$P(\mathcal{Y}\|\mathcal{X})$	posterior probability
$P(\mathcal{X})$	covariate probability
$P(\mathcal{Y})$	prior probability
$P(\mathcal{X}\|\mathcal{Y})$	covariate conditional probability
n_r	number of drifts
$n_{r'}$	number of signaled detections
r	set of drift locations
r'	set of detection locations
$\mathcal{MF}$	metafeature values for data chunks
e	limit of the classifiers in ensemble
$\mathcal{S}$	problem subspaces
n_s	size of the subspace
θ	threshold for drift detection
δ	sensitivity of drift detector
λ	minimal concept length in chunks
$\mathcal{PE}$	prior probability estimation method

Contents

Chapter 1

Introduction

The recent years have been referred to as the third summer of Artificial Intelligence (AI) [107, 109]. This field, identified earlier with cybernetics and connectionism [83], from its emergence in the 1940s, considered the data analysis, including the synthesis of knowledge on its basis [164].

Machine learning, one of the pillars of artificial intelligence, focuses on the construction of *intelligent systems*, capable of *generalizing* an abstract knowledge from provided data samples [9]. Machine learning systems can apply a knowledge model optimized for the considered task as a substitute for a manual algorithm definition.

The beginning of *the first wave* of artificial intelligence can be associated with the effects of theoretical work by McCulloch and Pitts [162], and later with the application and implementation of such system [194]. The research at that time was closely related to *cybernetics* – the study field aiming to understand the intelligence of animals and machines using control theory [234]. Much attention was committed to studying the feedback loop between the real-world environment and the system, allowing for the system's *learning*. At such an early stage of AI, the response to any modification of the intelligent model was delayed in time [109], since the selection of its parameters, in search for intelligent behavior, simulated the evolution process [67].

The suspension of work on artificial intelligence methods is referred to as the winter of AI. One can suppose that the cessation of funds preceding the first winter resulted in too high expectations from the premature field, also limited by the low computational resources [109]. Others may identify the publication of *Perceptrons: An Introduction to Computational Geometry* [168], revealing the limitations of intelligent machines as a direct cause of a paradigm shift for public opinion.

The second wave of artificial intelligence is associated, among others, with the development of expert systems. The first solution of this kind was presented in 1968 [59], but it was not until 1980 that they gained significant interest and funding [109]. Unlike in the first wave, financing expert systems was mainly driven by commercial solutions, hence, by a real market needs.

In parallel with expert systems based on logical rules, the *connectionism* movement was developing [197], in which researchers again focused their aims on studying human cognition. The breakthrough was associated with assembling individual units called *neurons* into layered networks, which allowed for witnessing intelligent behavior when such units cooperate [83]. The neurons assembled into hidden layers were believed to describe complex characteristics resulting from the interaction of simple elements visible in the system's input [197]. During the second wave of AI, another critical aspect of today's *deep learning* was proposed – the *backpropagation* [198]. It allowed for optimizing the intelligent system's parameters using a *stochastic gradient descent*, calculated based on errors made by the method, replacing the evolutionary approach to learning seen in the first wave of artificial intelligence. The groundbreaking works published at that time include Neocognitron [68], introducing the foundations of today's *convolutional neural networks* (CNN).

Many concepts critical for today's AI directly resulted from the *connectionism* research in the second wave. Meanwhile, the development related to connectionism and expert systems also slowed down. Similarly to the first winter of artificial intelligence, investor expectations were not met, leading to reduced funding [83]. From today's perspective, a significant limitation was the computational costs of processing large datasets, needed to optimize intelligent systems. Expert systems, meanwhile, proved to be expensive to maintain and often suffered from the lack of trust among human experts [109].

Currently, we are witnessing *a third wave* of AI rising since around 2006 [83]. The development of the field is fueled by large amounts of generated, stored, and processed data, as well as access to computing power that enables the training of *deep learning* methods [9].

The circumstance that significantly drove the development of deep learning in the third wave was the *ImageNet* challenge [199], describing the classification task of over 1.2 million images from a thousand classes. This challenge showed how quickly deep learning, particularly *computer vision*, can develop. The groundbreaking work was the proposal of the *AlexNet* model in 2012 [130], using the implementation of convolutional neural networks on *Graphic Processing Units*, which allowed for efficient computations. In later years, the challenge resulted in the proposition of CNNs with an increasing number of parameters and increasingly complex designs of connections between units.

In 2015, the *ResNet* model [93] achieved the lowest error rate, surpassing the human-level performance [92].

In recent years, the attention of society and scientists has shifted partially away from *computer vision* tasks, which were already being solved by machine learning algorithms with high quality, and focused on *Natural Language Processing* (NLP), and in particular, on studying *Large Language Models* (LLM) [247].

When writing this dissertation, one can suppose that the third wave of artificial intelligence development has been significantly impacted by the works on LLM, expanding the interest in the research. While some scientists claim that LLMs are capable of *metacognition* – understanding the reasoning process [52] – society is seeking the signs of the genuine intelligence in these systems [37]. The rapid development in NLP fuels discussions about *trustworthy artificial intelligence*, its ethics and responsibility, as well as their impact on the environment [13]. Those aspects do not go unnoticed by corporations and the world's media [77], which further fuels the interest in artificial intelligence, often preying on society's fears [16, 220].

In the face of the perceived vulnerabilities of deep learning, canonical machine learning methods and solutions returning to the paradigms of *symbolic artificial intelligence* are being developed in parallel [96], focusing on explainability [245] and responsibility of AI.

Despite the persistent grand promises of AI solutions and a vibrant stream of funding similar to those preceding the beginning of the first and second winters of artificial intelligence, some researchers suppose that the next one will not arrive [109]. Still, considering the previous rises and falls of research in this field, the probable winter may eventually shift the research direction to a yet undiscovered path. Regardless of what the future holds, the research community can work on increasing the chances of continuous and gradual development of the *machine learning* research.

1.1 Machine learning

Since the very beginning of the research in *machine learning*, the solutions were designed to *learn* specific tasks based on provided data examples – first by applying the evolution-like feedback loop, and later, through complex mechanisms such as *backpropagation* [198]. In all of the developed approaches, the learning process was possible due to the knowledge *generalization* – searching for specific regularities present in the available data examples [169] – with the *knowledge* stating an effect of data analysis by the dedicated model.

1.1.1 Task taxonomy

Machine learning algorithms are abstractions that can be applied to various types of data to perform various *tasks*. Computer implementations of methods impose certain constraints related to, for example, the numerical representation of data and the specified precision of described characteristics. However, the domain described by the available *dataset* remains arbitrary. As the field was developing, many solutions were proposed that can be applied to the specific *task* under consideration.

The primary taxonomic axis of machine learning is the division into *supervised* and *unsupervised* learning [136], as shown in Figure 1.1. In supervised tasks, the goal of the procedure is to imitate the knowledge of a human expert, provided in the form of *labels* for available data *samples*. The method aims to discover the regularities to match the provided labels.

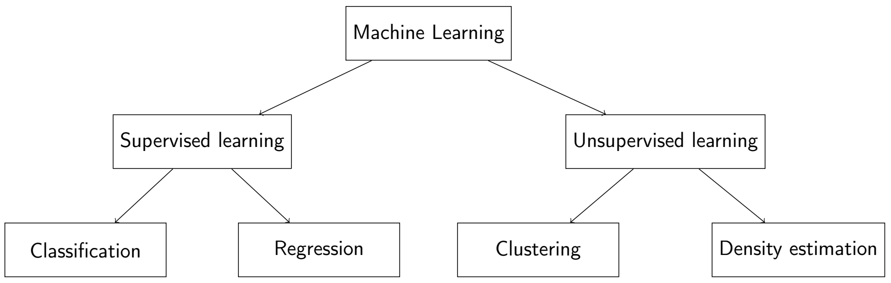

Figure 1.1: *The taxonomy of machine learning tasks. The primary axis of the field is the division into supervised and unsupervised learning. The most important tasks of supervised learning are classification and regression, while for unsupervised learning – the clustering and density estimation tasks.*

Following many real-world applications of machine learning systems in medicine, an example of *label*, describing the samples in a supervised scenario, could be a binary identifier describing if the person (sample) requires medical support. Applying a recognition system that solves this task could lower the facility's expenses by not relying on the time of qualified medical doctors. A trained machine learning system could automatically refer the patient to seek medical help based on already available labeled historical samples.

The described system performs an example of *classification*, i.e., a supervised learning task in which the label is of a discrete category, and is denoted as *class*. The classes, therefore, in the specific task described above, could be semantically understood as *requires medical support* (a positive case) and *does not require medical support* (a negative case). All data describing patient characteristics other than a *label* are denoted as *features* or *attributes*.

The second important task of supervised learning is regression, where the *label* describing a sample is of continuous type. Based on the available historical data, the system performing regression task could determine how long the person will spend in the facility from the moment of arrival to the moment of discharge. Hence, the particular difference between classification and regression is the type of *label* describing the data samples and, therefore, the type of the output of the recognition system.

Supervised tasks have gained significant interest from companies in commercial applications, as well as from researchers proposing the methods [207]. Machine learning systems solving supervised tasks allow a reduction in the working time of qualified employees, such as medical doctors [76], which possibly fuels the research on supervised approaches. However, it is worth remembering that the costs of expert support in supervised tasks are not entirely eliminated. The applications of such recognition systems, especially in complex domains [218], require a vast pool of valuable labeled examples provided by qualified experts [63]. It should also be remembered that algorithms generalizing knowledge based on labeled samples try to imitate the regularities concealed in labels. For this reason, any biases and errors visible in the labeled data are propagated to the recognition model. Such biases have been noticed in many solutions, highlighting the discrimination, bias, and errors of human annotators [61].

Unsupervised tasks form the second major branch of machine learning. In these tasks, the data examples are described only by the *attributes*, such as patient characteristics and illness symptoms – without the *label* describing the target of a recognition system. In unsupervised learning, the two largest groups of performed tasks are *clustering* and *density estimation* [47].

Clustering allows the grouping of examples into similar *clusters*. Like in classification tasks, the output of the algorithm is of discrete type – presenting, for example, an index or an identifier of the group. The clustering algorithm could group patients with similar symptoms into specific wards. Meanwhile, the task of *density estimation* aims to approximate the distribution density of the provided data samples. Such a task could allow for estimating the probability of a single symptom occurring in a given group of patients or – analogously to the regression task – allow for estimating the probability distribution of the time spent in the medical facility, assuming such data attribute is already available in the database. While *clustering* returns the discrete value, the *density estimation* method results in a continuous one, describing the probability of a studied event.

The mentioned task is related to the particular type of classification – *one-class classification* – where the objects of a single class are used to train the model. The model then assigns the particular area of problem space to the available category of events. Despite the *classification* problem being solved, such a task can be viewed as an unsupervised task of *density estimation*, where a model returns a value describing the support of class assignment to each input sample in the inference process. Both density estimators and one-class classifiers often use *Radial Basis Function* (RBF) kernel to approximate the probability distribution of data examples by analyzing their distances.

Unsupervised tasks do not require labels, meaning their employment often generates lower costs than supervised tasks. The effect provided by the model – for example, a clustering algorithm – may, however, deviate from the assumptions of the system creators, taking into account different features than anticipated or weighting them not according to the presumptions [101]. In extreme cases, if the patient's features include information about gender, ethnicity, and auxiliary factors like weight and height, the clustering system could potentially utilize such features in the generalization process, treating them as equivalent to the illness symptoms or case severity. As an effect – similarly to the *supervised learning*, where the *labels* required particular attention – the recognition system creator should be aware of the dependencies between the input and output of the particular solution and pay special attention to the data supplied to the system, in order to prevent abuse and errors resulting from its usage.

Finally, as a fusion of *unsupervised* and *supervised* learning, one can also observe the development of *semi-supervised* methods and *active learning*, in which the goal of the recognition system is to mimic the recognition of a human expert, but only some objects are marked with labels [204]. In such systems, the machine learning algorithm can interact with an outside world – for example, with a medical professional – to obtain labels for *uncertain* cases. The applications of *semi-supervised* and *active* solutions allow for significant reductions in the cost of obtaining labels [248] while maintaining the advantages of *supervised* tasks.

1.1.2 Classification

Classification task remains the most frequently considered one in machine learning research [127]. As initially stated in Section 1.1.1, the goal of the classification task is to assign a given sample to predefined discrete categories, denoted as classes, based on regularities found in examples labeled by a human expert [240], or analogously, to assign a discrete class label to a specific sample.

Equation (1.1) formally defines the sample x.

$$x = \{x^1, x^2, \dots, x^d\} \tag{1.1}$$

Each sample x is a d dimensional feature vector, where the consecutive features x^k come from feature space $\mathcal{X}$, as denoted by Equation (1.2). The k indicates a k^{th} feature, and the *dimensionality* d, the number of features describing each sample.

$$x \in \mathcal{X} = \{\mathcal{X}^1, \mathcal{X}^2, \ldots, \mathcal{X}^d\} \tag{1.2}$$

A *sample*, also denoted as a *pattern*, is an abstract representing any object in the recognition domain. Such an object is described by a set of *features* or *attributes*, which take the form of a vector.

A sample in the medical context can be viewed as a person or, more specifically, in the digital environment, as data describing the characteristics of a person. The *features* must be specified in a structured order, common for all available examples. This constraint indicates that the k^{th} feature for all available samples must describe specific characteristics of a person. The uniform structure of a processed dataset is necessary for the recognition algorithm to search for regularities in the provided data.

There are some applications where samples may be defined as data structures other than a simple vector [82]. The representation in the form of a graph may be utilized in NLP tasks. In the case of such applications, the text or acoustic data that is naturally understood by humans needs to be structured to allow for processing in a digital environment. Pre-processing such modalities may require, for example, identifying parts of a sentence to present a text in a graph form [165]. It is worth emphasizing, however, that the default representation for most machine learning tasks is a vector. For the presented NLP example, *vectorization* techniques are often utilized, which construct a sample representation by analyzing the word frequencies and their order or semantics [131]. In the case of another structured data format – like images in *computer vision* tasks – the various *feature extraction* methods are used to generate vector representation of samples, typically by analyzing the affinity and the neighborhood of image pixels [157].

The significant benefit of supervised machine learning approaches is the possibility of applying the available knowledge for new data samples instead of relying on manual recognition. First, the classification method acquires abstract knowledge in the *training* or *learning* process to later *infer* for a sample without a given label. The generalization is performed based on the *training set*. The new samples (not present in the training set) come from the real-world environment, where the system is applied, or, in the case of evaluation of machine learning methods, from the *test set*.

Equation (1.3) formally describes the classification function Ψ performed by a classification algorithm. The *classes* semantically indicate the discrete categories present in the recognition task and are identified by *labels* $\mathcal{Y} = \{y_1, y_2, \ldots, y_M\}$ [240].

$$\Psi : \mathcal{X} \to \mathcal{Y} \tag{1.3}$$

A classification function assigns one of the labels y coming from a set of labels $\mathcal{Y}$ based on the sample's features x from feature space $\mathcal{X}$. This process is described by Equation (1.4).

$$\Psi(x) = y_k \text{ if } x \in X_k \tag{1.4}$$

The label y_k is assigned if the sample x belongs to a decision area X_k (a component of $\mathcal{X}$). This process is described in Equation (1.5).

$$\mathcal{X} = \bigcup_{k=1}^{M} X_k \text{ where } (\forall\ y_1, y_2 \in \mathcal{Y})(y_1 \neq y_2 \implies X_{y_1} \cap X_{y_2} = \emptyset) \tag{1.5}$$

The equation describes the feature space $\mathcal{X}$ as a union of decision areas X_k, where the regions are associated with labels. As presented in the equation, the decision areas are disjoint – hence, the mapping $\mathcal{X} \to \mathcal{Y}$ is unequivocal.

The large number of features describing a given task makes the recognition task more complex – since the generalization in many dimensions becomes increasingly difficult. This is referred to as the *curse of dimensionality* [227]. Canonically, dimensionality reduction techniques are used to counteract such increasing complexity of generalization. These include *feature selection* and *feature extraction* [111]. In feature selection, the characteristics seen as the most informative are selected, for example using statistical tests. Meanwhile, in the extraction, the complete set of features is transformed to generate a reduced number of new attributes, resulting from the projection of the original feature space. One of the methods performing a feature extraction is *Principal Component Analysis* (PCA) [34, 221] – which returns the requested number of most informative components of the data.

The decision areas X_k are determined and defined based on the samples used to train the classification model. Depending on the specific algorithm, there are various approaches to defining the decision areas embedded into the classification function. A simple example may be *k-Nearest Neighbors Classifier* (KNN), where the decision area is defined based on the label of k nearest instances. A different way of generalization can be seen in the *Decision Tree Classifier* (DT), which uses splitting criteria to *slice* the available feature space into specific class regions.

The example of the decision areas defined by KNN and DT classifiers for the two-dimensional feature space and synthetically generated training samples are presented in Figure 1.2.

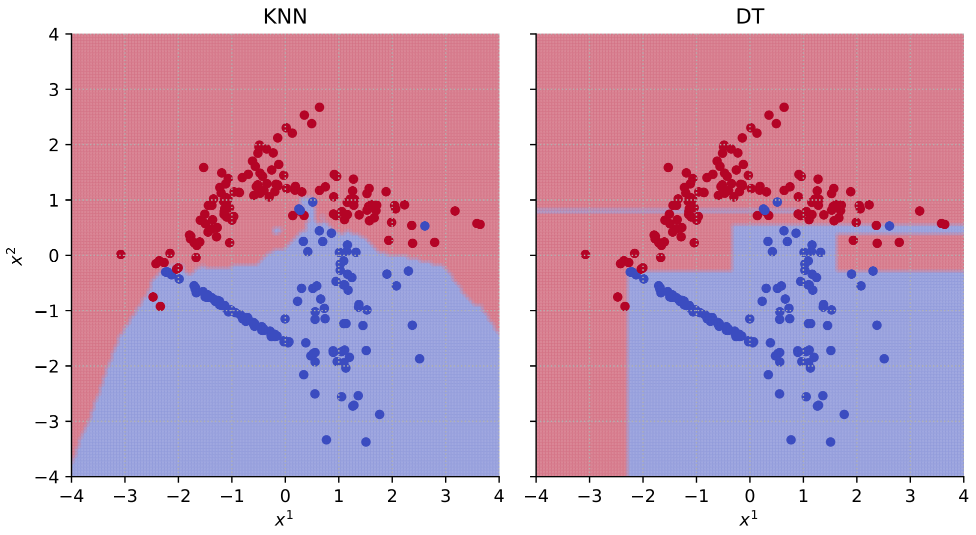

Figure 1.2: *The decision areas of k-Nearest Neighbor Classifier (with k=5) and Decision Tree Classifier, fitted to the available synthetic data samples. The colors indicate the specific decision areas, while the points indicate the available training examples.*

Formally, the figure presents a binary classification problem, where $\mathcal{Y} = \{0, 1\}$ and the *dimensionality* of a feature space $\mathcal{X}$ is $d = 2$. The class samples identified with label $y = 0$ are denoted as blue points, while samples of the class whose label $y = 1$ are shown in red. The decision areas were identified using the specific algorithms. The regions identified by different colors indicate the decision areas for given classes. When a new sample (for example, coming from a testing set) belongs to a specific area, the classification algorithm Ψ assigns a relevant label to such sample. The abstract line, or in more general, high-dimensional feature spaces, a *hyperplane*, separating the decision areas, is denoted as a *decision boundary* [240].

This figure allows to observe the effects of the model's *generalization* of the available *evidence*. The definition of class regions depends on the underlying algorithm, and therefore, various classifiers can be more or less suitable for a specific application of a machine learning system. This idea is related to as *no free lunch* – a theorem first described by Wolpert for optimization tasks [236]. It is also often applied to supervised machine learning [2], where it states that there is no universal solution to the possible problems [240]. What follows is that various classification methods perform differently for various tasks, which makes the correct and detailed evaluation of machine learning problems of critical value.

Even if the model is perfectly fitted to the available training data samples – meaning that all available training examples are correctly classified – it may produce errors for data unseen in the training process. An extreme example of such a phenomenon is known as *overfitting*. It is especially visible in methods capable of defining complex decision boundaries and trained over many iterations, such as neural networks (NN) [18]. Overfitting results in the loss of the generalization ability of the model. When such a phenomenon occurs, the complex model *memorizes* the training examples – creating complex decision boundaries, where the mislabeled examples (or examples characterized by noise) often excessively distort the decision areas. The example of overfitting for more complex training data has been presented in Figure 1.3 for a *Multilayer Perceptron* (MLP) [197] – a neural network implementation with fully connected hidden layers.

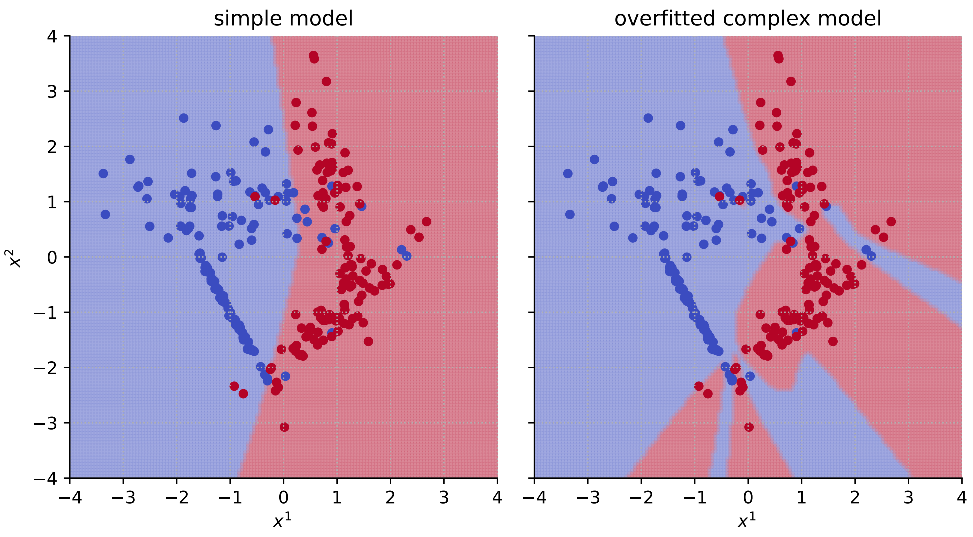

Figure 1.3: *The decision areas defined by the Multilayer Perceptron model with different hyperparameters – on the left with a single hidden layer and 100 training iterations, while on the right with five hidden layers and 1000 training iterations. The example shows the* overfitting *phenomena, typical for models capable of producing complex decision boundaries.*

The left side of the figure presents the decision areas defined using a model with a single hidden layer, trained over a hundred iterations. The right side of the figure presents an output of the classifier with five hidden layers, trained over a thousand iterations fitted to the exact data. Those variables describe the *hyperparameters* of the model – the values specified by the user that determine the internal operation of the learning algorithm – how the algorithm selects its *parameters*. In the case of MLP, *hyperparameters* describe the sizes of hidden layers and the loss function, meanwhile *parameters* – the values of weights and bias of individual units, optimized during a learning process.

To mimic the real-world scenarios, the synthetic data presented in Figure 1.3 was additionally characterized by a label noise [246]. As a result of overfitting, the model optimized the decision areas to correctly classify the mislabeled examples, resulting in a very complex decision boundary, precisely fitted to the training data – but not necessarily for the data used to test the method's performance or to samples describing the *real-world* application, for which the model was optimized. The fact that using the same underlying algorithm (MLP) with different hyperparametrization and different training times resulted in such different decision areas for the same training data shows that algorithm selection in machine learning problems is a complex task.

It is worth mentioning here that most of the classifiers, apart from a discrete decision, provide some form of a probabilistic *support function*, describing the *certainty* of a decision. In the case of the *k-Nearest Neighbors* classifier, this support is defined according to the number of neighbors of each class – the more neighbors of a specific class, the higher the probability of a sample belonging to this class. In the case of *Multilayer Perceptron*, and neural networks in general, those *supports* are calculated based on the output from the last layer of a network structure. The outputs of neural network nodes are also denoted as *activations*. After passing through each hidden layer, those are subjected to the *activation function*, which determines the values of weights in a model. At a final layer, in the case of classification tasks, those activations are typically followed by the *softmax* function, defining a probability of a sample belonging to each class present in the recognition task. Figure 1.4 presents the example of support function values for two mentioned classification algorithms – KNN and MLP.

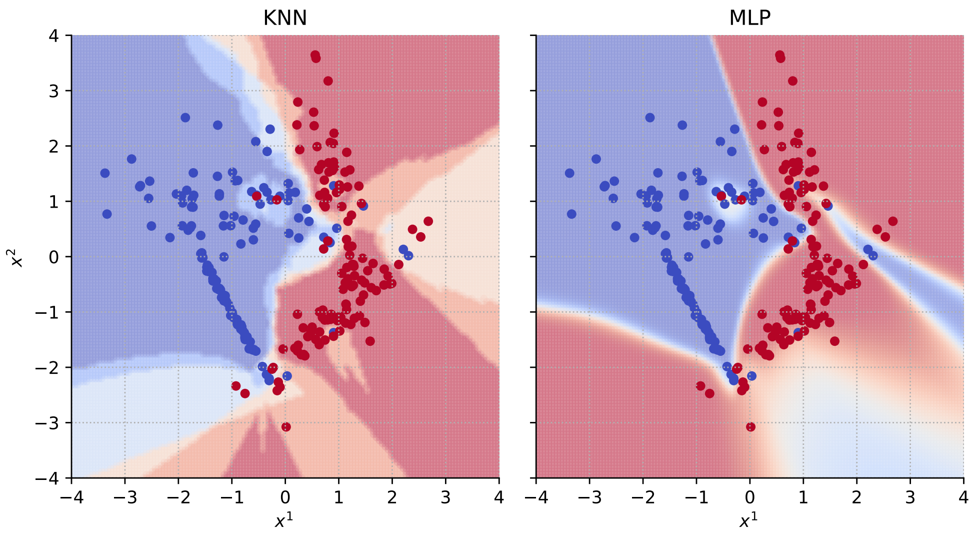

Figure 1.4: *The feature space of a synthetic classification problem, marked according to the classifier's probability support functions for a binary classification task. The left side of the figure presents the support function of the* KNN *model with $k = 5$, and the right side – the support function of the* MLP *model with a single hidden layer.*

The regions of feature space described with saturated red color correspond to a high support towards a positive class ($y = 1$) – interpreted as a high probability of a sample belonging to a positive class – while saturated blue color describes the high support towards the negative class ($y = 0$). The less saturated colors describe *uncertain* regions – that lack a clear and distinct identification of a given class. As can be seen in the figure, the regions close to the possibly mislabeled samples – outliers of a specific class – are often characterized with lower *certainty.* The same applies to the regions between clusters of different classes, close to the *decision boundary.* The final decision of a classifier is calculated according to the support of a classifier – meaning choosing a more probable label for *uncertain* samples.

Besides the canonical classification, the *uncertainty* in general can be used in the tasks of semi-supervised and active learning, where – as mentioned in the Subsection 1.1.1 – the system can request labels for the ambiguous samples whose supports towards more than one class are high. The lack of *explainability* of neural network predictions is often visible when examining the activations of a model. Especially in the cases of high-dimensional data and unsupervised feature extraction typical for deep learning methods, neural networks tend to classify potentially *uncertain* or even *unknown* [201] samples with high support – what was noticed as a relevant vulnerability of such models [219].

1.1.3 Metafeatures and metalearning

Generalization as a mechanism of a classification algorithm becomes an essential factor of a learning process [9]. As shown in Section 1.1.2, available algorithms (or even the same algorithm with different configuration) can generalize available evidence in various ways, ultimately presenting different decision boundaries and dividing the decision space into abstract categories. Often, the main objective of machine learning is to select an appropriate processing pipeline, along with an appropriate classification algorithm [240].

The promising direction allowing for the appropriate classifier selection may be *metalearning* [30] – a branch of machine learning whose main objective is to aid the appropriate processing pipeline definition. *Metalearning*, at a general level, can be understood as *learning about learning.* The solutions try to *generalize* how certain characteristics of the available data influence the learning process. Such characteristics are defined as *metafeatures* – informally understood as *features of the features* – describing the sample's attributes at a more general, dataset- or batch-level. One can say that *features* are the characteristics describing the *sample*, while *metafeatures* are the characteristics that describe the *dataset.*

Metafeatures are critical to metalearning, as based on their values, the promising solutions are selected [192]. They are then most often used to predict the quality of classification for a given algorithm [192] but can also be used for other tasks, including the description of datasets for evaluation [28] or the construction of benchmark datasets [171].

One can present an *imbalance ratio* as a highly critical characteristic of the dataset considered in the classification task. As described in the Subsection 1.1.2, the discrete categories of instances in the mentioned tasks are denoted as classes. Some problems are characterized by a disproportion in the cardinality of specific class instances. This is denoted as *imbalance* and causes difficulties at the level of recognition model optimization and proper method evaluation. Most classification methods are characterized by *bias*, directly resulting from the underlying *generalization* process. As a consequence, models tend to present a preference towards the majority class – the more numerous one. The problem of *class imbalance* is typical for medical diagnostics. Often, in such scenarios, the task of the algorithm is to classify samples into categories describing a healthy individual (negative class) or an individual with a specific disease (positive class) – with the disease being relatively rarely observed in the general population and therefore, in the available data used to train the model [62, 112].

A basic example of the *k-Nearest Neighbors* classifier can be given here. In the case of significant imbalance, the required number of k samples of a specific class may not be present in the training set, which could result in assigning an entire feature space to a majority class. An *imbalance ratio* treated as a metafeature can be used to select the appropriate k hyperparameter of the classifier, or, what is more typically done in case of imbalanced data, to use a pre-processing of the available samples in the form of resampling techniques [112]. Such a pre-computation of the dataset's metafeature can significantly impact the development of the processing pipeline in the recognition system, benefiting the overall model recognition quality.

Formally *class imbalance* as a metafeature – denoted as $C2$ [154] – is one of the measures describing the *complexity* of the classification problem. The direct imbalance ratio (IR) can be calculated based on the proportion of each class cardinality n_k divided by a cardinality of opposite class samples $n - n_k$, as presented in Equation (1.6). Then, the IR is used to calculate $C2$, as shown in Equation (1.7).

$$IR = \frac{M-1}{M} \sum_{k=1}^{M} \frac{n_k}{n - n_k} \tag{1.6}$$

$$C2 = 1 - \frac{1}{IR} \tag{1.7}$$

The $C2$ has been defined in such a way that its values are limited to the range from 0 to 1. The balanced problems provide a low complexity of 0, while problems with a high class imbalance – the values close to 1.

The example of balanced data and data characterized by a class imbalance is presented in Figure 1.5. The example on the left side of the figure presents a balanced 2-dimensional binary classification problem. In contrast, the example on the right presents a problem where the positive class samples (red points) are less frequent than the negative class samples (blue points).

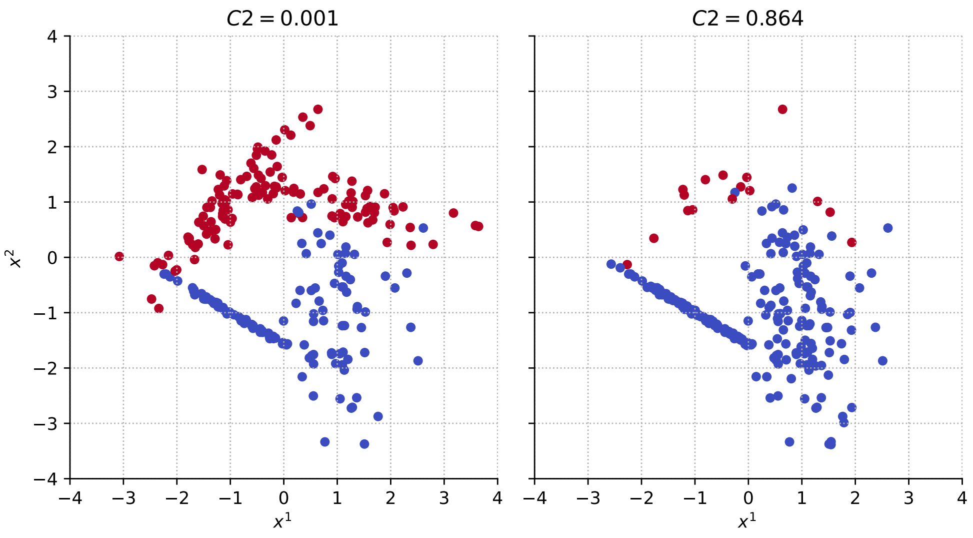

Figure 1.5: *The examples of datasets characterized by different class proportions: the balanced problem (left) and the imbalanced one (right). Red points indicate positive class samples, while blue points – a negative class. The measure of $C2$ presented in the subfigure titles aims to describe the complexity of the data.*

In the description of the examples in the figure, the value of $C2$ metafeature is presented. According to the semantics of complexity measures, the imbalanced dataset presented on the right states a more complex challenge regarding the classification task. It is worth noting that class imbalance is among the simple metafeatures that could be easily assessed visually by examining the sample distribution or by simply calculating the class frequencies. However, the range of metafeatures that are possible to compute is relatively wide – ranging from simple ones, such as *dimensionality* or *imbalance ratio*, to significantly more complex that require the use of heuristic solutions [97] or the use of classifiers like *k-Nearest Neighbors* [154] or *Decision Trees* [208].

A vast number of *complexity measures* that aim to assess the difficulty of the supervised recognition tasks has been proposed by many researchers [97] and aggregated in the works by Lorena et al. [154, 155]. Such measures for classification tasks describe, for example, the separability of the problem classes, the neighborhood of samples, and the area of class overlap in the feature space. The *complexity measures* form a taxonomic branch of metafeatures and have been successfully used to describe the characteristics of the datasets that translate to the performance of the recognition models [117].

Metafeatures of a dataset are formally defined as the function that returns values characterizing the set of samples. The metafeature for supervised machine learning tasks was formally defined by Rivolli et al. [192]. Each sample in the training set, apart from feature vector x, is additionally described by a label y. The available dataset $\mathcal{D}$ consisting of n instances is therefore described by Equation (1.8).

$$\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \dots (x_n, y_n)\} \tag{1.8}$$

The metafeature f is defined as a function $f\colon \mathcal{D} \to \mathbb{R}^p$, such that when applied to the dataset $\mathcal{D}$, it returns p values describing the given set. The definition of a metafeature is presented by Equation (1.9).

$$f(\mathcal{D}) = \sigma(m(\mathcal{D})) \tag{1.9}$$

The following components were used to define a metafeature:

- $m\colon \mathcal{D} \to \mathbb{R}^{p'}$ – a characterization measure,
- $\sigma\colon \mathbb{R}^{p'} \to \mathbb{R}^p$ – a summarization function.

A summarization function is used when a specific cardinality of p is required, and the number of p' differs from the expected one. If the $p = p'$, the functions m (characterization measure) and f (metafeature) may be consistent, and σ is the identity function.

The σ function is used when m returns more values than the application expects (p'). This may happen when analyzing pairs of attributes of the original dataset or calculating values for individual features, while the single value $p = 1$ characterizing the dataset is required. Then, the σ often returns the mean value of the computed p' metafeatures. Alternatively, other statistical determinants can be used to extend the context of the processed data [190].

1.1.4 Data stream processing

The complexity measures, briefly described in Subsection 1.1.3, aim to assess the difficulty of processed tasks. Such measures, along with many other metafeatures, are primarily used to describe *static* datasets that do not arrive over time. There exists, however, another data structure and, therefore, another category of machine learning problems characterized by different types of difficulties. The potentially *infinite* data inflows in the form of *streaming* data – *stationary* or *non-stationary* – arriving over an extended time. The *non-stationary* data states a particular challenge, as the data distribution evolves over the processing time [3].

Data streams are typically utilized when processing large volumes of data [74], especially in applications where they are generated and delivered in real-time and an almost immediate response or the recognition system is required [69]. Such applications are highly characteristic of a modern digital world, driven by real-time data generation and processing.

The data stream $\mathcal{DS}$ is formally described as a set of samples x, as shown in Equation (1.10). One of the difficulties related to this type of data is the possibility of infinite data arrival. Therefore, machine learning systems must be prepared for continuous data processing without increasing their computational complexity.

$$\mathcal{DS} = \{x_1, x_2, x_3, \ldots\} \tag{1.10}$$

In the case of supervised tasks, in addition to the feature vector x, the data stream contains labels as shown in Equation (1.11).

$$\mathcal{DS} = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots\} \tag{1.11}$$

With the above assumption, however, it should be remembered that the label may not be available immediately but can be delivered after some time. This problem is reflected, for example, in the *Test-then-train* experimental protocol, in which first the features x of processed instances arrive, and inference is performed using the evaluated recognition method. Labels y for processed samples arrive in the next iteration, and only then are used to *iteratively* train the recognition model [23].

The continuous training of the machine learning algorithm is not always possible by design. For example, in the case of the *k-Nearest Neighbors* classifier, which stores the training dataset in the model memory, storing an increasing number of samples is theoretically possible. However, with the assumption of infinite sample arrival – the underlying algorithm requires certain modifications to allow for effective data stream processing. Such an example aims to show that incremental learning of machine learning methods, necessary for *non-static* data, forms a challenge and often requires adaptations of original approaches suitable for static data.

The response to the inability of canonical recognition methods to effectively process data streams has fueled the evolution of *ensemble approaches* applied to this type of data [128].

Ensemble methods, integrating multiple classifiers (or other recognition methods), each adapted to the specific sub-task, are often used to process data streams thanks to the possibility of dynamic adjustments of the ensemble structure or its parameters.

The definitions of data stream presented so far describe the case of stream processing in an *online* form – when the stream consists of individually arriving samples. A slightly different approach to processing data streams is *offline* or *batch* processing – analyzing data portions (denoted as batches or chunks). Such a data stream is described in Equation (1.12).

$$\mathcal{DS} = \{\mathcal{DS}_1, \mathcal{DS}_2, \mathcal{DS}_3 \ldots\} \tag{1.12}$$

The stream that arrives in batches consists of data chunks $\mathcal{DS}_k$ of given size n. Each chunk contains n problem instances, as shown in Equation (1.13). The data stream itself still, by definition, remains infinite, but the samples are aggregated into uniform-size subsets. In such a scenario, the batches of data can arrive infinitely.

$$\mathcal{DS}_k = \{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)\} \tag{1.13}$$

It is worth mentioning here that any data stream in an *online* form can also be processed in batches when there is a possibility of storing the data in a buffer of size n. Therefore, the subsequent data batches may constitute separate subsets within which the *metafeatures* can be calculated. Monitoring the values of metafeatures over the data stream processing can extend the benefits of using them in static datasets.

One can describe a data stream processing scenario where the objective is to estimate the time spent in the medical facility based on the patient's characteristics – which, as mentioned in Subsection 1.1.1, is a canonical regression task. As typical for real-world applications, the system is first trained on a batch of labeled examples and later induces knowledge for new examples over the system's operating time. If the method stores the *features* describing new patients, after some time, system owners acquire the *actual* labels, describing the real time spent in the facility. Such valuable data can form a new *data batch* and be later used to incrementally train the recognition system.

By optimistically assuming that the described phenomenon is *stationary*, there may be a specific point in time when the system does not require further training. However, the *dynamic* and *non-stationary* real-world applications are characterized by ubiquitous changes. For instance, a particular disease may require less treatment time due to a new drug proposal. Therefore, the recognition system processing the data stream should exhibit the ability to adapt to changes in the processed data according to the provided labels.

The described variability in the context of data streams is denoted as *concept drifts*, and data streams characterized with concept changes – as *non-stationary.* The non-stationarity of the data forms a final challenge related to data stream processing.

1.1.5 Concept drift

The *concept* has been referred to since the early works in the area of artificial intelligence [169]. The term *concept* was then used to describe a given class or category of instances [164]. The objective of machine learning was described as *concept learning* and viewed as a search problem. As Mitchell stated in his dissertation: *the ability to generalize from specific instances of a class to a general model or description of a class, [is] often referred to as concept learning* [169].

In modern machine learning, the idea of a *concept* has gained a probabilistic interpretation. It has been formally defined in the literature as the *joint probability* $P(\mathcal{X}, \mathcal{Y})$, dependent on the *posterior probability* $P(\mathcal{Y}|\mathcal{X})$, and the *covariate probability* $P(\mathcal{X})$ [71, 110]. Equation (1.14) gives the relation between these two components and the joint probability.

$$P(\mathcal{X}, \mathcal{Y}) = P(\mathcal{Y}|\mathcal{X}) \times P(\mathcal{X}) \tag{1.14}$$

The *posterior probability* is further defined based on *covariate probability* $P(\mathcal{X})$, *prior probability* $P(\mathcal{Y})$ and *covariate conditional probability* $P(\mathcal{X}|\mathcal{Y})$ [128], as indicated in Equation (1.15)

$$P(\mathcal{Y}|\mathcal{X}) = \frac{P(\mathcal{Y}) \times P(\mathcal{X}|\mathcal{Y})}{P(\mathcal{X})} \tag{1.15}$$

Specifically in the case of data streams, where there is a time factor, probabilities are defined for time instants $P_t(\mathcal{X}, \mathcal{Y})$ [74]. This time factor becomes crucial when formally describing the changes happening over time. Concept drift occurs if the joint probabilities $P(\mathcal{X}, \mathcal{Y})$ at two time instants t and u differ. This relationship is described by Equation (1.16).

$$P_t(\mathcal{X}, \mathcal{Y}) \neq P_u(\mathcal{X}, \mathcal{Y}) \tag{1.16}$$

An extreme example of a concept drift, described as *pure class drift*, could be seen if the *covariate probability* $P(\mathcal{X})$ remained static, while only *posterior probability* $P(\mathcal{Y}|\mathcal{X})$ changed over time, affecting the decision areas of the recognized problem. Such a change is not possible to detect in the case of unsupervised data processing – when only *covariate probability* $P(\mathcal{X})$ is observed. In most of the real-world scenarios, however, it is both the *covariate probability* $P(\mathcal{X})$ and the *posterior probability* $P(\mathcal{Y}|\mathcal{X})$ that are impacted due to a concept change [128].

The probabilistic definition of concept drift is aligned with the original understanding of the concept as a *generalization of a class*, presented by Mitchell [169] – in the face of a concept change, the definition of a category in the machine learning task changes, hence, the abstract knowledge describing a given problem evolves over time.

In the area of data stream processing, the *concept* does informally describe the data distribution and the semantics of classes. *Concept drift* occurs when data distribution or the definition of classes changes over time. The traditional *static* datasets can be viewed as describing a single, *stationary* concept, while, in the case of incremental processing of data stream, the data can change over time [50] – forming a *non-stationary* data stream. From the perspective of machine learning systems, concept drift is an important phenomenon, as it usually leads to a loss of recognition quality in the implemented system [180].

The example of a concept drift and its impact on the recognition quality is presented in Figure 1.6. The top row presents the chunks of data at different time intervals, denoted as *chunks*. The visualization utilized a synthetically generated stream where, over

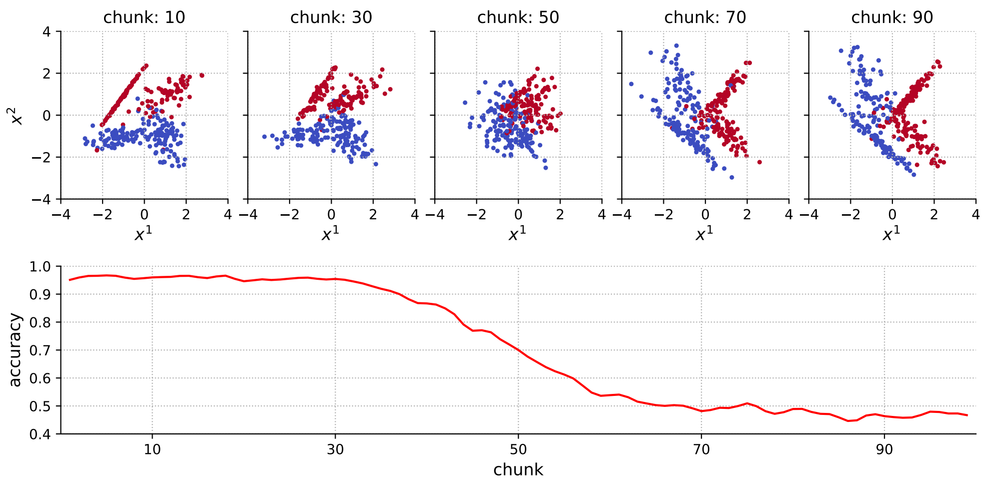

Figure 1.6: *An example of a concept drift in the synthetically generated data stream, processed in the form of batches. The distribution of data samples in specific chunks is shown in the top row, while the classification accuracy of a non-adapted classifier – in the bottom row. The colors of points indicate the class of an instance.*

a hundred chunks, a single concept drift occurred. The bottom row shows the classification accuracy of a *k-Nearest Neighbor Classifier*, fitted on the first data chunk and not updated throughout the data flow.

It can be first noticed that the sample distribution, presented in the top row, changes over the processing time – which is especially visible when comparing the data distribution from the 10^{th} and 90^{th} data chunks. The accuracy curve over the processing time shows that, if the system does not respond to changes in the data distribution, the recognition quality continues to decline or even, as presented in the example scenario, the classifier entirely loses the recognition ability by achieving the accuracy of 50% – which is equivalent of a random prediction in the binary classification problem.

By following the example presented in Subsection 1.1.4, the system that estimates the patient recovery time will present poor recognition quality if, for example, new treatment options are invented, new infection outbreaks or any real-world event impacts the operation of a medical facility. More generally, concept drift may occur as a result of any seasonal and daily cycles [98], the emergence and disappearance of trends [183], outbreaks of the spread of infectious diseases [222] or any dynamic evolution of recognized task, extremely characteristic of real-world applications [231].

A typical occurrence for applications where seasonal cycles are observed is the *concept recurrence*. In such applications, the specific concepts interchange over the processing time. This scenario offers a possibility not only to respond to the concept changes and incrementally train the model – typical for non-recurring changes – but also to propose solutions that do not require a complete learning of a concept already known from the past.

While metafeatures are usually used to describe a static dataset [30], they can also be used in the data stream setting, even though the entire dataset never arrives. Such a possibility is offered by processing the data stream in the form of batches. While the metafeatures designed for static data can also benefit methods for data stream processing [87], Webb et al. proposed *quantitive measures* specifically for the data streams [230, 231]. Those measures, directly dedicated to describing a concept drift, can be used as metafeatures calculated over subsequent batches or moving windows accumulating the arriving samples over a specific time.

The motivation behind those quantitive measures was to objectively describe the changes visible in the data [231], allowing for the *detection* of a concept change that does not rely on the consequences of concept drift (i.e., the drop of recognition quality) – but examines the data distribution itself [230]. The task of *concept drift detection*, stating a vital area of research, places nowadays a significant impact on the *implicit* drift detection, whose mechanism does not rely on monitoring the quality of underlying classification model [85].

Webb et al. [230, 231] proposed three measures used to describe the concept drift phenomena for the classification task. The measures compare the sample distributions from two time instants t and u and are defined as follows:

- *Drift Magnitude* (DM) measures the distance between two concepts $P_t(\mathcal{X})$ and $P_u(\mathcal{X})$. The original measure uses the *Hellinger distance* [20] and is described as:

$$DM_{(t,u)} = \frac{1}{\sqrt{2}} \sqrt{\sum_{x \in \mathcal{X}} (P_t(x) - P_u(x))^2}. \tag{1.17}$$

- *Conditioned Cardinal Covariate Drift* (CMCD) is defined as the weighted sum of the distances between conditioned covariate distribution $P(\mathcal{X}|\mathcal{Y})$ for each problem class. The weights are the average prior probability of occurrence of a given class $P(\mathcal{Y})$ at both points in time. The measure is using the *Total Variation Distance* [145] and is described as:

$$CMCD_{(t,u)} = \sum_{y \in \mathcal{Y}} \left[\frac{P_t(y) + P_u(y)}{2} \cdot \frac{1}{2} \sum_{x \in \mathcal{X}} |P_t(x|y) - P_u(x|y)| \right]. \tag{1.18}$$

- *Posterior Drift* (PD) is defined similarly to CMCD as a weighted sum of distances but uses covariate distributions $P(\mathcal{X})$ and posterior distribution $P(\mathcal{Y}|\mathcal{X})$. The measure is using the *Total Variation Distance* and is described as:

$$PD_{(t,u)} = \sum_{x \in \mathcal{X}} \left[\frac{P_t(x) + P_u(x)}{2} \cdot \frac{1}{2} \sum_{y \in \mathcal{Y}} |P_t(y|x) - P_u(y|x)| \right]. \tag{1.19}$$

In the case of using these measures, batch processing of data is necessary to aggregate a set of samples that reliably describes the processed data distribution. Later, the estimation of the probability density is performed.

The DM measure describes *total* drift magnitude, while CMCD and PD the *marginal* drift magnitude. The measure of DM examines the *covariate probability* $P(\mathcal{X})$, whereas the following measures also utilize *prior class probability* $P(\mathcal{Y})$, *posterior probability* $P(\mathcal{Y}|\mathcal{X})$ and the *conditioned covariate probability* $P(\mathcal{X}|\mathcal{Y})$, offering a more complex understanding of processed data.

The *concept drift detection* and the *non-stationary data classification* play a significant role in this dissertation. As an inherent characteristic of *dynamic* real-world applications, concept drifts should be considered in any data stream processing system [158].

1.2 Research hypothesis

The following dissertation addresses the issues related to the processing of *non-stationary data streams*, proposing solutions for the tasks of *concept drift detection* and *data stream classification*. The proposed solutions analyze the *metafeatures* calculated in disjoint data batches of a potentially infinitely incoming data stream. The research hypothesis is as follows:

> *It is possible to propose methods employing metafeature analysis for concept drift detection and classification of the non-stationary data streams that demonstrate significantly better or statistically dependent recognition quality compared to state-of-the-art approaches.*

Aims and goals In order to substantiate the above hypothesis, the following objectives were formulated, dedicated to the tasks of *concept drift detection* and *classification* of *non-stationary data streams*:

1. Proposal of an *implicit concept drift detector* analyzing the time variability of the *classification task complexity measures* calculated for disjoint data chunks. The completion of this objective is described in Section 3.1.

2. Proposal of an *implicit concept drift detector* analyzing *drift magnitude measures* integrated using the *ensemble learning paradigm*. The completion of this objective is described in Section 3.2.

3. Proposal of an *unsupervised drift detection method* analyzing the distribution of *activations* from the last layer of a *deterministic neural network*. The completion of this objective is described in Section 3.3.

4. Proposal of an *ensemble method* for *classification of data streams* analyzing the distributions of *statistical metafeatures* calculated for subsequent data chunks to identify recurring concepts. The completion of this objective is described in Section 4.1.

5. Proposal of a *classification* method for compensating the bias of baseline classifiers when processing the data streams with dynamic changes in *imbalance ratio*, using the prior probability estimated based on the *metafeatures* of the processed data chunk. The completion of this objective is described in Section 4.2.

6. Proposal of a framework for processing data streams with a time-varying level of difficulty, allowing for the selection of an appropriate neural network architecture based on the analysis of the model's *certainty*. The completion of this objective is described in Section 4.3.

1.3 Structure of the dissertation

The dissertation has been divided into five chapters.

Chapter 1 discussed the fundamental subjects related to the field of machine learning and the undertaken research topics, including *data stream processing*, *classification*, and *metafeature analysis*. The introduction of these concepts allowed to determine the motivation, research hypothesis, and detailed goals of the planned research, presented at the end of the first chapter.

Chapter 2 presents the related works and background essential to this dissertation in more detail. It describes the motivation of *metalearning* and key aspects of the data stream processing. Subsequently, it introduces the *state-of-the-art* approaches to the *concept drift detection* and *data stream classification*, expanding on the correct and reliable experimental evaluation of data stream processing methods solving these two tasks.

Chapters 3 and 4 present the developed methods for the tasks of *concept drift detection* and *data stream classification*, respectively. Each of these chapters presents three methods employing the analysis of *metafeatures* calculated over batches of a data stream to solve the undertaken machine learning task. A definition and presentation of each proposed method are followed by a description of the experimental environment and an analysis of the obtained results.

Chapter 5 summarizes the work presented in this dissertation and highlights potential future directions of the research.

Chapter 2

Related works and background

This chapter expands on the fundamental topics of machine learning introduced in the first chapter that are critical to the research presented in this dissertation. The existing processing strategies, methods, and related works are discussed in more detail following the current literature of the subject.

To comprehensibly supplement the basic concepts presented in the Chapter 1, each section begins with a reference to topics already discussed in the Introduction and includes information about the extensions contained in a particular section.

The second chapter has been divided into four sections. The first one aims to describe the *taxonomy of metafeatures*, primarily describing the two groups significant for the presented research. The second section focuses on general aspects of *data stream processing* – expanding on the limitations of such systems and presenting the data stream sources. The next section describes the *concept drift detection task* – focusing on the taxonomy of concept changes and presenting the *state-of-the-art* techniques of concept drift detection. The last section presents the area of *data stream classification*, describing the existing approaches.

The last two sections were supplemented with a description of topics related to the evaluation of data stream processing methods, including the processing modules and evaluation criteria specific to the considered tasks.

2.1 Metalearning and metafeatures

The intuitive understanding of metafeatures and their formal definition has already been presented in the Subsection 1.1.3. So far, the primary goals of metalearning were presented, along with an example of employing a simple metafeature describing the imbalance ratio of a classification problem. Moreover, the complexity measures – a family of metafeatures important for the research presented in this dissertation – were briefly described.

This section expands on the goals of metalearning, presenting its target development paths and identified constraints. Later, the section describes the current taxonomy of metafeatures, presenting its categories and characterizing the two most important ones for the research presented in this dissertation.

Metalearning studies the learning process of the machine learning methods [144]. The main goal of this research area is to determine what data characteristics affect the recognition quality and to describe such impact [226]. Most metalearning systems are nowadays used for algorithm recommendation, which allows for quick and automatic selection of methods based on their performance in similar scenarios [192].

A first metalearning system described by Rice studied a similar problem of *algorithm selection* [191]. In the mentioned work, author defined the *problem space* and *algorithm space*. The former contains many independent determinants (characteristics) of the problem that are important for the algorithm selection – from a modern perspective, those could be viewed as metafeatures. The latter defines a set of algorithms and their configurations that are considered for the problem. Therefore, the *problem space* could be interpreted as metafeature space, and *algorithm space* as the possible set of targets, with each class describing the particular algorithm best suited for a specific problem.

2.1.1 The fundamentals of metalearning

Formally, metalearning considers the *metaproblem*, in which *metafeatures* describe the *metatarget* [60]. The metafeatures are the properties describing the given dataset, and the metatarget can be any value that the system aims to learn. In recommendation systems, metatarget can be defined as the recognition quality using a given algorithm or a discrete label indicating the suitable algorithm or its parameters [30]. The scheme of such abstraction is presented in Figure 2.1 – showing how, based on dataset $\mathcal{D}$, the metafeature and metatarget spaces are defined.

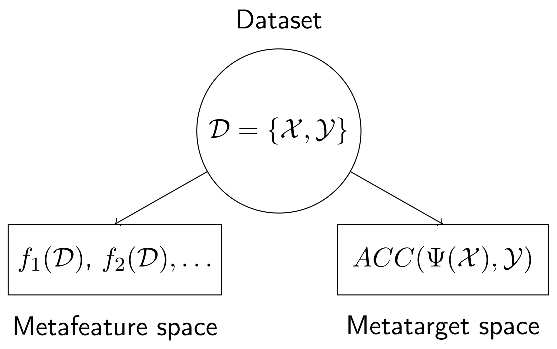
Dataset
$\mathcal{D} = \{\mathcal{X}, \mathcal{Y}\}$
$f_1(\mathcal{D}),\ f_2(\mathcal{D}), \ldots$
$ACC(\Psi(\mathcal{X}), \mathcal{Y})$
Metafeature space
Metatarget space

Figure 2.1: *The scheme showing the metafeatures and metatarget derived from the dataset. The metafeature space shown of the right consists of various measures describing the dataset $\mathcal{D}$. The metatarget space shown on the right consists of a continuous classification accuracy value with a given model Ψ applied to the dataset $\mathcal{D}$.*

From the perspective of canonical machine learning, a recommendation system can perform a regression task – where the target variable is of continuous type and describes the classification quality of a specific algorithm measured with the selected metric. The features of such a new metalearning problem are characteristics of a dataset – defined as *metafeatures* $f_1(\mathcal{D}),\ f_2(\mathcal{D}), \ldots$ describing the data on a general level. To prepare such a metalearning recommendation system, it is necessary to collect a dataset describing the metaproblem. Such a superimposed dataset consists of factors describing the original data – e.g. imbalance ratio, problem dimensionality, or the measures describing the data sparsity – and the continuous label describing a recognition quality using a given method – in the presented example $ACC(\Psi(\mathcal{X}), \mathcal{Y})$.

When the acquired dataset describing a metaproblem is large enough, one can train a canonical regression model to predict the recognition quality depending on the dataset characteristics. Such a process has been illustrated in Figure 2.2, where the metafeatures of numerous datasets $\mathcal{D}_1,\ \mathcal{D}_2, \ldots$ along with their corresponding recognition accuracy with a specific classifier Ψ have been used to train a metalearning model Ψ_M.

Such a solution, or comparable solutions that directly recommend an algorithm as a discrete label, can facilitate and speed up the development of a recognition system. The techniques originating from metalearning are successfully used in *AutoML* [94], where the objective is to automatically select a method and its hyperparameters for a given problem. The use of metalearning could allow for obtaining the appropriate algorithm and its configuration without thoroughly evaluating a vast pool of available algorithms and their hyperparameters.

It is worth remembering that the metafeatures describing a new metalearning problem provide some generalized data characteristics [45]. Meanwhile, the relation between the problem and the recognition quality can be very complex. The selected metafeatures

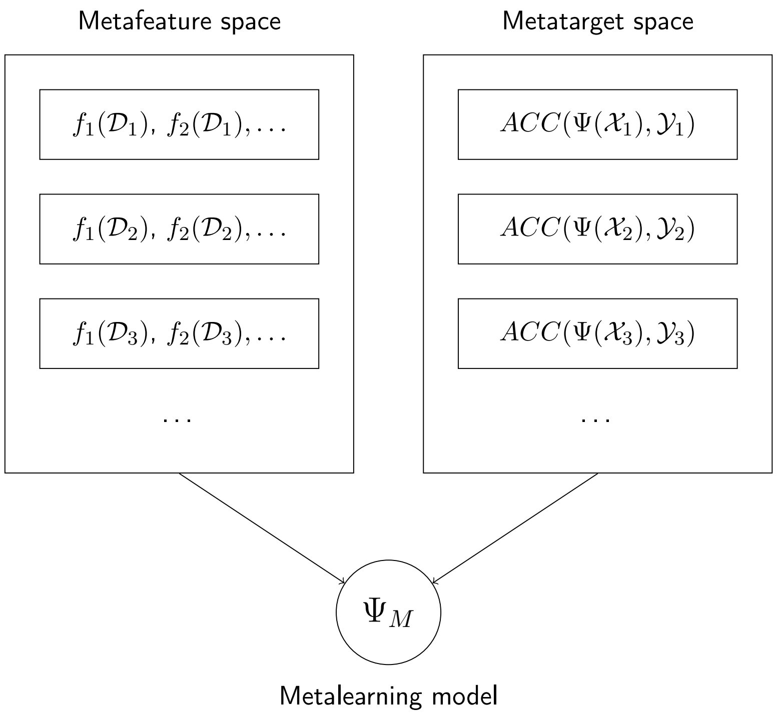
Metafeature space
Metatarget space
$f_1(\mathcal{D}_1),\ f_2(\mathcal{D}_1), \ldots$
$f_1(\mathcal{D}_2),\ f_2(\mathcal{D}_2), \ldots$
$f_1(\mathcal{D}_3),\ f_2(\mathcal{D}_3), \ldots$
$\ldots$
$ACC(\Psi(\mathcal{X}_1), \mathcal{Y}_1)$
$ACC(\Psi(\mathcal{X}_2), \mathcal{Y}_2)$
$ACC(\Psi(\mathcal{X}_3), \mathcal{Y}_3)$
$\ldots$
Ψ_M
Metalearning model

Figure 2.2: *The scheme showing the construction of a metalearning model Ψ_M, which is trained using the set of metafeatures for a large number of datasets and the corresponding metatargets.*

may not provide enough information about the dataset, or the provided characteristics can be of little significance concerning the algorithm's performance. Moreover, a wide range of problems is required to enable metaknowledge generalization by a metalearning system. A considered case of supervised learning enforces not only the calculation of metafeatures for such problems but also the acquisition of metatarget, which, in recommendation systems, often requires the complete training (to the point of model convergence) of many machine learning solutions only to examine their quality. These aspects are viewed as current limitations of metalearning and *AutoML* [108].

While one of machine learning objectives is choosing the best processing pipeline for the particular problem, or even the development of a principal, *master* recognition algorithm [53], scientists are placing their hopes on metalearning techniques despite the currently identified challenges [228].

2.1.2 Metafeature taxonomy

Over the years, a range of measures and metafeature taxonomies have been considered, aiming to adequately describe the input data for thorough problem understanding.

The most extensive description and analysis of available metafeatures are currently available in work by Rivolli et al. [192]. The authors systematized the available measures and assigned them to six main categories:

Simple describe the basic properties of a dataset and are the simplest in terms of both definition and computational complexity.

Statistical designed for numerical attributes, intended to describe the statistical properties of a data distribution.

Information theoretic operate on categorical attributes. They are intended to describe the amount of information contained in the data based on entropy.

Model based describe the properties of a decision tree fitted to the available data.

Landmarking describe a dataset based on the quality obtained for fast and simple learning models.

Others include measures with higher computational cost or domain bias, divided into four additional categories [7]:

> **Clustering** examine data after subjecting it to clustering with a selected algorithm or use class labels as cluster identifiers.
>
> **Complexity** aim to describe the difficulty of the classification task.
>
> **Concept** describe the irregularity and consistency of the data.
>
> **Itemset** analyze the binary categorical attributes. They provide information about the frequency of an attribute and about the correlations between their pairs.

The following paragraphs focus on two metafeature categories most critical for the presented research – *statistical* and *complexity* groups.

Statistical metafeatures Metafeatures from the statistical category describe the wide range of statistical properties of data. The examples include the maximum, minimum, average, or correlation values of problem features. This section presents the formal example of metafeature calculation, following the definition provided in Chapter 1.

Among these metafeatures, values are often calculated within individual features of a d-dimensional dataset (such as the mean or maximum) or within pairs – each feature with the target value (such as the correlation). In the example of the *maximum*, the characterization measure m is a function calculating the maximum value within each attribute of the dataset $\mathcal{D}$ consisting of n samples, which results in obtaining $p' = d$ metafeatures.

The function characterizing each feature $j = 1, 2, \ldots, d$ is described by Equation (2.1), presenting the calculation of a *maximum* statistical metafeature. The described dimensionality relation is described by Equation (2.2).

$$m^j(\mathcal{D}) = max(x^j) \tag{2.1}$$

$$m : \mathcal{D} \rightarrow \mathbb{R}^d \tag{2.2}$$

If the goal is to obtain a single value characterizing the given dataset $p = 1$, using the summarization function σ is necessary. Such a function can be described, for example, as the *mean* value of the obtained characterization measures. Equation (2.3) describes the summarization function calculating the mean value of initial metafeatures. The dimensionality of the metafeature after applying this summarization function is equal to $p = 1$, as shown in Equation (2.4).

$$\sigma(m^1, m^2, \ldots, m^{p'}) = \frac{\Sigma_{j=1}^{p'}(m^j)}{p'} \tag{2.3}$$

$$\sigma : \mathbb{R}^{p'} \rightarrow \mathbb{R}^1 \tag{2.4}$$

The category of statistical metafeatures also includes complex measures, such as the number of normally distributed attributes, the number of outliers, and measures based on the results of Multivariate Analysis of Variance – for example, *Pillai's Trace* or *Wilks' Lambda* [29].

Complexity measures The complexity measures category is of particular importance in this dissertation. The measures were described and systematized in detail by Lorena et al. [154]. They differ in definition from those described in the publication by Rivolli et al. [192] due to the inclusion of a summarization function by design, which means that each of the described measures results in a single metafeature value ($p = 1$). The list of metafeatures describing the complexity and their importance within the categories is presented in Table 2.1.

According to the available literature, the scope of data complexity measures as problem metafeatures is relatively wide, ranging from applications in signal data [147], through the dominant application in metalearning [192], extending the context of difficult data

Table 2.1: *The description of complexity measures groups – the name of a category, their description and the list of measures that belong to the specific category.*

Measure category	Semantics of measures	List of measures
Feature-based	The measures describe the ability of features to separate classes in the classification problem. They analyze features separately or evaluate how attributes work together.	F1, F1v, F2, F3, F4
Linearity	The measures evaluate the level of problem class linear separation. They use the Linear Support Vector Machines (SVM) classifier to define linear decision boundary between the classes.	L1, L2, L3
Neighborhood	The measures analyze the neighborhood of instances in a feature space. Neighbors of each sample are established based on the distance between problem instances.	N1, N2, N3, N4, T1, LSC
Network	The measures consider the instances as the vertices of the graph. All measures of this category utilize an *epsilon-Nearest Neighbors* graph, where the edge is placed between the points if a normalized *Gower distance* is smaller than 0.15. Edges between instances of distinct classes are removed.	Density, ClsCoef, Hubs
Dimensionality	The measures analyze the relation between the number of features and the number of instances in the dataset.	T2, T3, T4
Class imbalance	The measures evaluate the dataset based on the class cardinality proportions.	C1, C2

analysis [78], or applications in supporting the classification models of imbalanced data classification [14].

In imbalanced problems, they allow estimating the difficulty of data, which was confirmed by experiments on synthetic data [15]. There is also a noticeable correlation between the classification quality and the problem complexity measures when using the oversampling algorithms [117].

2.2 Data streams

The data stream was formally defined in the Subsection 1.1.4. The data streams are considered difficult data, as the methods for their processing are restricted with certain limitations. An important factor in data stream processing is the concept drift. The informal understanding of a concept and its formal probabilistic definition was already presented in the Subsection 1.1.5, additionally showing an example of concept drift effects on the data distribution and classification quality.

This section describes the fundamental guidelines of data stream processing systems in more detail, considering the restrictions of physical computer systems and the widespread occurrence of concept drift. The description of such changes is expanded according to the current taxonomy of the phenomena. Further, this section describes the sources of data streams used in computer experiments, discussing the benefits and drawbacks of their selection.

Data streams are increasingly often employed in many machine learning applications [71], often aiding the topical tasks of *big data* analysis. *Big data* is characterized by large *variety*, *velocity* and *volume* [41, 200]. When considering those three characteristics, data streams are especially beneficial in conquering the challenges related to its *volume* and *velocity*. The data volume does not allow storing the samples in physical memory and makes incremental processing necessary [3]. Moreover, almost all real-world applications with a high velocity of data inflow require an immediate system response and appropriate adaptation to changing environments. Examples of such applications include network traffic monitoring systems [115], text message processing [172] and web click analytics [106], as well as *internet of things* applications [181].

2.2.1 Processing requirements and limitations

The keys to effective data stream processing are real-time analysis and the lack of historical data accumulation. Algorithms should be capable of continuous processing without consuming excessive memory resources. These limitations were thoroughly described by Domingos et al. [55], forming a set of criteria that should be met in order to process data streams efficiently:

- *A little processing time per sample is required.*

  As the data streams are usually utilized in applications with a high data velocity, the system response should be almost immediate. The lack of such behavior can cause a system to respond when the particular response is no longer needed. An example could be the positive signaling of a fraud detection system, processing the payment information as a data stream – responding to a particular query with a time delay – once a transaction has already been accepted. From a user's point of view, such a delayed response is of little value.

- *The consumed memory resources need to be independent of the number of previously processed samples.*

  The data streams are defined as potentially infinite, meaning the entire dataset never arrives. Keeping in mind that some recognition methods suitable for processing static data store the artifacts of training samples in the method's memory (an extreme example is the *k-Nearest Neighbors* classifier, which stores the entire training set), certain modifications should be made to allow for infinite processing of data. One of the approaches is to limit the number of data samples or the method's components stored in the process memory. The use of non-adapted methods increases the computational complexity of the system with a processing time, which, in turn, impairs the first constraint or entirely violates the recognition system.

- *The need for a single-time processing of each sample.*

  This constraint expands the previous limitation. It enforces that the samples are not excessively stored in the model's memory. In the case of online processing, the samples should be discarded once they have been used to train the model. In the case of batch processing, the same requirement is applied to a data chunk. However, some generalized knowledge can be stored in the form of the algorithm's components – such as the location of class centroids or probability parameters describing the distribution of available classes – which is done by the Gaussian Naive Bayes (GNB) classifier, naturally adapted to incremental training.

- *The algorithm should be able to process data at any moment.*

  This restriction touches upon the system's reliability and is of significant value when considering the concept drifts. Given that such a concept change is detected, some steps are usually performed to prevent the recognition quality drop – e.g. the classifier is rebuilt. Such steps should remain transparent from the user's perspective and not cause a time delay in a system response. This restriction also enforces the system's reliability, which is consistent with previous guidelines. A system not well adapted to incremental data processing could stop its operation due to an internal failure.

- *Any limitations resulting from the processing of the data stream should not affect the recognition quality.*

  As mentioned, the solutions dedicated to data stream processing often require certain modifications and compromises. Any such adjustment should not cause the method to perform worse than static data processing. Since the data streams form a difficult problem, this constrained may not be always possible to fulfill. However, the researchers should aim to bring the recognition quality as close to the *static* baseline as possible, taking to account other limitations of data stream processing environment.

- *In the case of varying data characteristics, the model should always be up-to-date and contain relevant past information.*

  This guideline directly relates to concept drifts occurring in the data stream. The methods should adapt to a newly appearing data distribution. At the same time, in the case of recurring concepts, the model should be able to restore past knowledge. As the data stream is potentially infinite and the consecutive concepts may not be possible to predict, storing past knowledge for a long time is usually only possible with certain assumptions – such as setting a time limit after the old knowledge is discarded.

Data streams pose a challenge to canonical methods of machine learning, not adapted to the velocity of incoming data samples and time-changing domains [128]. The restrictions mentioned above enforce the specific mechanisms of data stream processing methods, such as forgetting the old data distributions. This supports the fact that the data streams constitute a challenging processing environment.

2.2.2 Concept drift taxonomy

As noted, some of the data stream processing guidelines directly refer to a concept drift. In real-world, open environments, there is an almost infinite number of types of changes that can occur in a data stream. In available sources, over the years, the concept drift has been described from various angles, depending on its source, dynamics, and effects of an occurring change [71, 166, 231].

As described in Section 1.1.5, the phenomenon of a concept drift is an effect of a change in the *joint probability* $P(\mathcal{X}, \mathcal{Y})$, which can translate into a change in its components: *covariate probability* $P(\mathcal{X})$, *prior probability* $P(\mathcal{Y})$ and *covariate conditional probability* $P(\mathcal{X}|\mathcal{Y})$. The taxonomy presented below first discusses the changes in *joint probability* and later describes the specific case of changes in the *prior probability*.

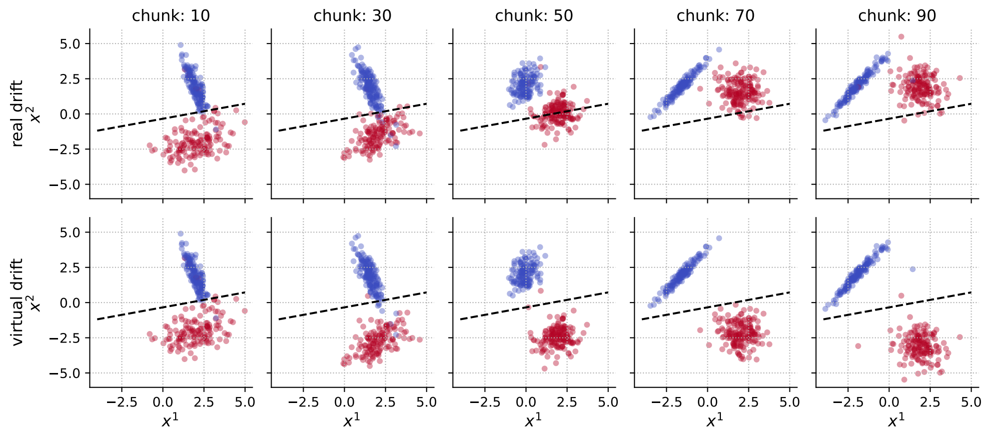

Figure 2.3: *Examples of real (top) and virtual (bottom) drifts. The distribution from the 10th chunk shows the initial concept, while the distribution from the 90th chunk – the subsequent one. The black dashed line indicates the linear decision boundary of a logistic regression model fitted to the data from the first concept. Compared to the real one, the virtual concept drift does not affect the decision boundary.*

Drifts in *joint probability* Following the currently established taxonomy [5], concept drift can be taxonomically divided according to three main axes:

- *Influence of the decision boundaries*

  The drifts can be categorized into *real* and *virtual* ones [4]. Those two types of drifts are presented in Figure 2.3. Despite the distribution shift in both real and virtual drift scenarios – the decision boundary defined for the initial concept allows for accurate classification of the following concept in the case of virtual change. Virtual drifts, therefore, are associated with changing data distribution without affecting the decision boundaries. Thus, they are not visible when assessing the classification quality of models. In contrast, the real drifts change the decision boundary and directly impact the recognition quality.

  It is worth noting, however, that virtual drifts can indicate the initial phase of real drifts with a concept change stretched over time. Early detection of such changes can positively impact recognition quality if appropriate steps are taken.

- *Drift Dynamics*

  Concept changes may occur at a single point in time or can be stretched over a more extended period [217]. The examples of each type of drift in this context are presented in Figure 2.4. In the case of *sudden* drifts, the concept change occurs over a shorter period than the data sampling. This results in a change

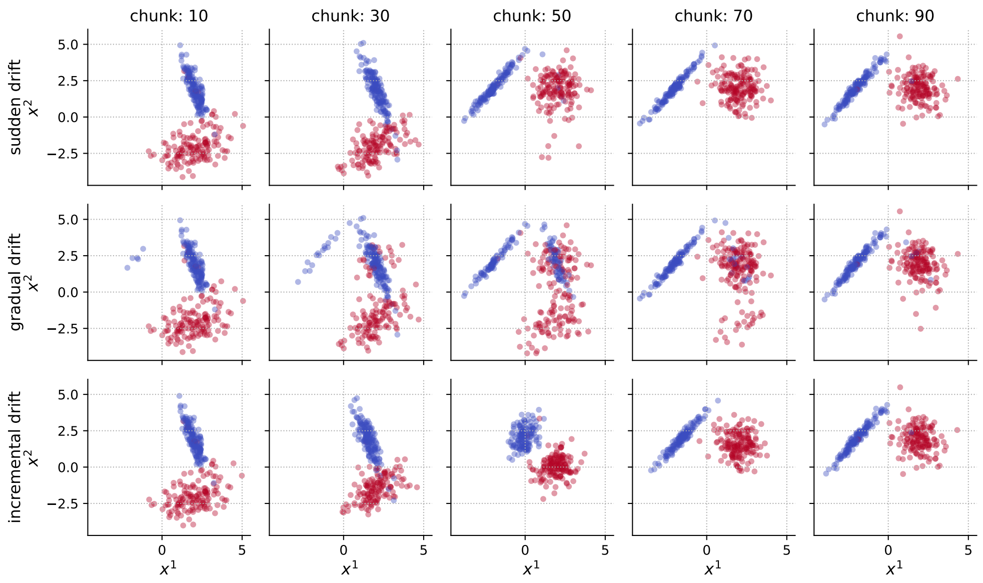

Figure 2.4: *Examples of sudden (top), gradual (center), and incremental (bottom) concept changes over the single concept transition. Sudden concept drift occurs at a single point in time, while gradual and incremental ones are stretched over time, with a distribution in a transitional phase depending on the type of drift.*

viewed as occurring at a single point in time. In the case of *incremental* and *gradual* drifts, the changes are smoother – occurring over a longer interval than data sampling, and therefore can be viewed as a smooth change from one concept to another. In the case of gradual drifts, samples from successive concepts appear in batches with varying probabilities. In contrast, in incremental drifts, there is a progressive transition in which samples belong to a temporary concept that depends on successive stationary concepts.

The temporary concept visible in the transition period of the gradual and incremental changes can sometimes be interpreted as an accessory one – containing either an overlap of samples from consecutive concepts (gradual drift) or the weighted average of features (incremental drift).

- *Concept recurrence*

A concept observed in the past may reappear after some time. While this characteristic describes the consecutive concepts, not the drifts themselves, it is often identified as an additional taxonomic category – represented as *recurring concept drifts* [217]. This type of drift can be characteristic of cyclic phenomena, such as seasons and daily cycles. Methods designed for recurring concepts often retain the classifiers used in previous concepts to re-utilize them when the concept reoccurs [89]. Figure 2.5 presents examples of non-recurring and recurring concepts.

In the case of the recurring concept, one of the data stream processing requirements presented by Domingos et al. [55] becomes critical. As mentioned, the data stream processing methods should adapt to a new concept but, at the same time, store the valuable information that could be used in the future when the concept reappears. The methods dedicated to recurring concept changes should, therefore,

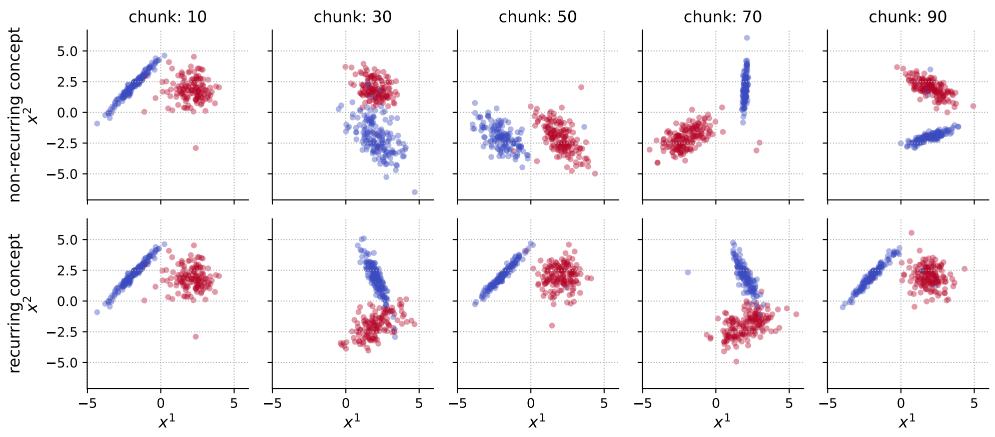

Figure 2.5: *Examples of non-recurring (top) and recurring (bottom) concepts over the data stream with five concept drifts in 100 data chunks. In the case of a non-recurring concept, after each drift, the new concept appears, whereas, in the case of a recurring one, the concepts alternate.*

retain some of the previous knowledge and be ready to use it when the concept reappears after some time. However, it is worth to note that there are other constraints related to the data stream processing – e.g. the maintenance of stable memory and computational complexity. This enforces the researchers to seek a compromise between the past knowledge accumulation and the efficiency of the methods, resulting in the upper bounds of the number of classifiers stored in many ensemble methods [128].

Drifts in *prior probability* Drifts in the *prior probability* constitute a specific type of concept drift that occurs in data streams related to the $P(\mathcal{Y})$ component of the *joint probability* $P(\mathcal{X}, \mathcal{Y})$. Such changes are related to the time-varying problem imbalance ratio [124].

The impact of varying *prior probabilities*, like varying *joint probability* distribution, can substantially impact classification quality [110]. Meanwhile, the strategies that can be applied due to significant class imbalance differ from those typical for changes in *conditional probabilities*, most often addressed in the research area of handling concept drift [195].

The concept drift that alters only the *prior probability* can be viewed as a unique case of *virtual* concept drift, in which the data distribution changes while the decision boundary remains constant. What is non-trivial about this type of change is the effect on the classification quality – as highly imbalanced problems usually constitute a challenge to canonical classifiers [5]. Therefore, one can view a virtual concept drift as a paradox, where the recognition quality can be affected [110], despite the lack of direct decision boundary shift. The selection of evaluation metrics for varying problem imbalance also

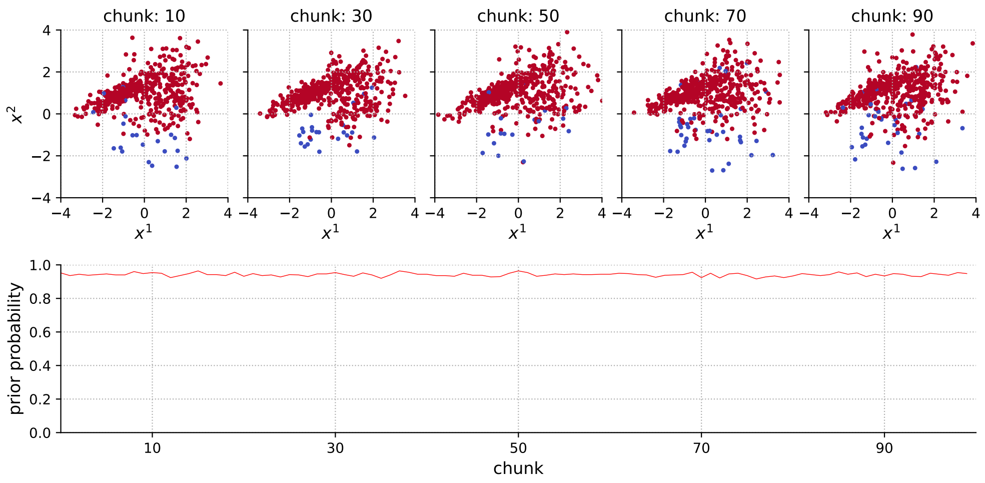

Figure 2.6: *A data distribution (top row) in a* statistically imbalanced data stream. *The bottom row shows the local prior probability of positive class $y = 1$, calculated for each data chunk. The prior probability remains at around 95% throughout the entire data stream processing time.*

affects how the assessed classification quality changes. The simple accuracy metric may not show the degeneration of the model's ability to recognize minority class instances, revealing the accuracy paradox [215]. Meanwhile, the metrics suited for imbalanced data evaluation most often degenerate over time in case of increasing class imbalance [105].

The literature on the subject highlights the problem of imbalanced streams [70] – often referred to as *skewed data streams* – and imbalanced streams in which additional, conditional probability drifts occur [75, 250]. The research in this dissertation touches upon the rarely addressed issue of data streams with time-varying *prior probability*, without the drift in *covariate conditional probability*. Taking the class imbalance into account, the data streams can be categorized into the following categories [124]:

- *Statically imbalanced streams* (SIS) – where the global prior and each of the local priors are characterized by a disproportionate but constant and dependent value. Figure 2.6 presents an example of such a data stream. In this taxonomic type, the level of imbalance is maintained at a similar level during processing.
- *Dynamically imbalanced streams* (DIS) – among which there are two subcategories:
  - *Continuous dynamically imbalanced streams* (CDIS) – where the global prior may differ from local priors, whose value is independent but changes continuously, allowing for the observation of trends in its changes. Figure 2.7 presents an example of such a data stream. In this data stream type, the imbalance ratio changes gradually over a stretched period.
  - *Discrete dynamically imbalanced streams* (DDIS) – where the global prior may differ from local priors, whose values are independent and change in a discrete

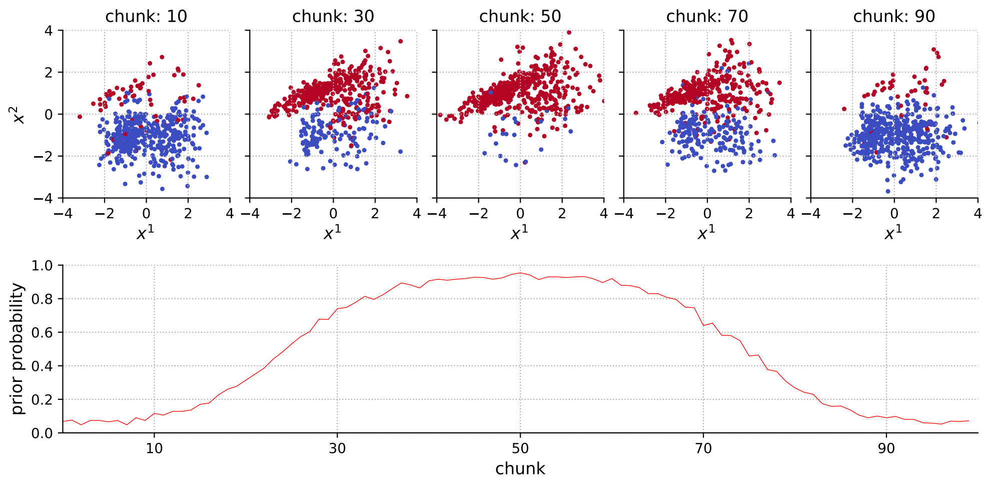

Figure 2.7: *A data distribution (top row) in a* continuous dynamically imbalanced data stream. *The bottom row shows the local prior probability of positive class $y = 1$, calculated for each data chunk. The prior probability changes from around* 5% *at the initial processing period to around* 95% *at chunk* 50.

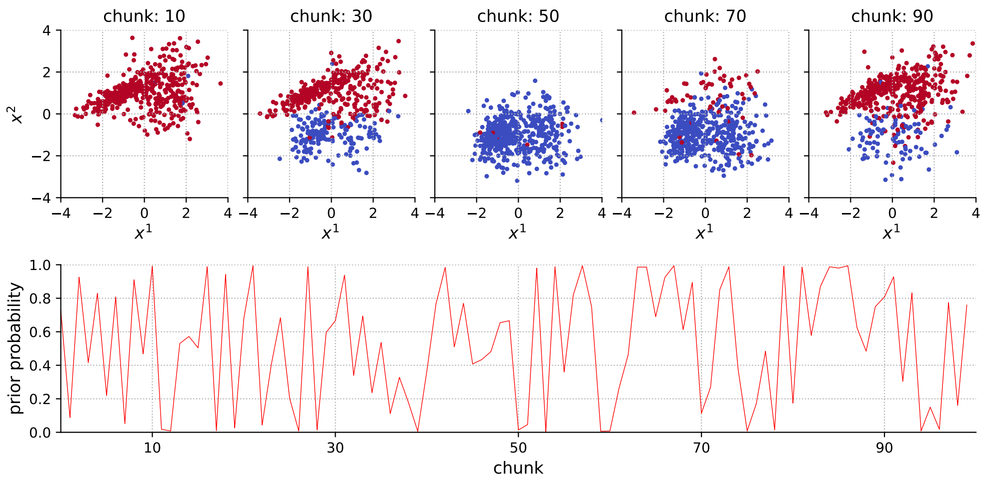

Figure 2.8: *A data distribution (top row) in a* discrete dynamically imbalanced data stream. *The bottom row shows the local prior probability of positive class $y = 1$, calculated for each data chunk. The prior probability changes discretely, with large deviations in consecutive data chunks.*

manner, making it impossible to observe trends in its changes. Figure 2.8 presents an example of such a stream. In the case of *discrete* changes, the local priors differ significantly from one chunk to another.

The *balanced streams* (BS) and data streams with a constant level of imbalance (SIS) have already received significant attention in the data stream classification research. Similarly, in the area of concept drift, much more attention is given to the drifts in the *joint probability*, and especially *covariate conditional probability* $P(\mathcal{X}|\mathcal{Y})$, rarely focusing on the drifts in *prior probability* $P(\mathcal{Y})$. The research presented in this dissertation fills this gap by considering various *sources* of concept drift – including both *covariate conditional* and *prior probability* – as well as studying only the changes in *prior*, without the drift in *covariate conditional probability*.

2.2.3 Data stream sources for computer experiments

The research related to data streams is usually carried out based on two types of data streams: synthetic or real-world [158]. Each of these types has its advantages and disadvantages that should be considered when performing a reliable experimental analysis.

Often, the emphasis is placed on analyzing real-world data [214], as it describes the authentic applications where machine learning systems are implemented and deployed. However, *real-world data stream* analysis has significant limitations in the context of the drift detection task. The main problem is the lack of accurate determinants of drift moments – denoted as *concept drift ground-truth* – which makes it impossible to compare methods based on metrics other than classification quality [153, 158]. What is more, the availability of real-world data streams is low [212]. The available streams can be too simple in terms of classification tasks, contain a small number of samples, or describe problems of low dimensionality.

For the above reasons, researchers often use *synthetic data streams* to properly assess recognition's effectiveness in changing environments. A significant advantage of this type of stream acquisition is the ability to specify stream parameters to obtain data with specific characteristics, such as the number of features, the number of drifts, and the length of the stream. Moreover, synthetically generated data can be replicated using various random seeds, allowing for extensive experimental evaluation and enabling statistical analysis of the results. Finally, synthetic data is easy to reproduce and can be explicitly regenerated instead of stored [23, 26].

One of the most significant benefits of using generators for non-stationary data stream acquisition is that they store precise markings of places where synthetically injected drifts occur. In sudden drifts, the exact moment of change is indicated, while in gradual or incremental changes – the precise function of a drift dynamics [133].

A third source of data streams combines the advantages of real-world and synthetic data streams. Based on widely available static datasets, it is possible to generate *semi-synthetic streams*, modifying the distributions of available data in order to simulate concept drifts [117] or sampling the available data in a specific way so that the data arriving over time are characterized by time-varying factors [123].

As in the case of any stream source, there are some disadvantages to using semi-synthetic data generators, which vary depending on the synthesis approach. Suppose the generator modifies the distributions of static datasets. In that case, the original semantics

Table 2.2: *The advantages and disadvantages of the specific data stream sources. The first two columns describe specific characteristics, while the final three columns indicate the category of data streams for which the characteristic is valid.*

Category	Characteristics	REAL	SEMI-SYN	SYN
Advantage	Describe real-world applications of machine learning systems	x		
	Describe *concept drift ground truth*		x	x
	Describe type of drift		x	x
	Possibility of replication		x	x
	Specify data stream characteristics (dimensionality, informativeness, number of classes)			x
Disadvantage	Low availability	x		
	Requires the data storage	x		
	The repeatability of sampled instances		x	

of the features is altered, for example, by using feature projections to enable the synthesis of concept drifts or to modify the dimensionality of the stream. In the other generation strategy – when the generators do not modify the original data from the static dataset but only sample them depending on specific criteria, multiple sampling of the same object is possible. The effect of such a procedure may be the execution of inference for samples that were previously used to train the model, which can happen if the classifier is incrementally training throughout the data stream processing.

Table 2.2 presents the considered characteristics of individual data stream sources, divided into advantageous and adverse ones in relation to the experimental analysis. The real-world data streams have a significant advantage of describing an actual application of a machine learning system. Meanwhile, as mentioned, they are of low availability and require data storage. Both semi-synthetic and synthetic data streams allow for the replication of a generation process and offer the *concept drift ground truth*, along with the description of the type of drift. Fully synthetic data streams, however, allow for more precise data specification compared to semi-synthetic data streams, which partially rely on static data.

Examples of data streams from each source have been presented in Figure 2.9. The rows

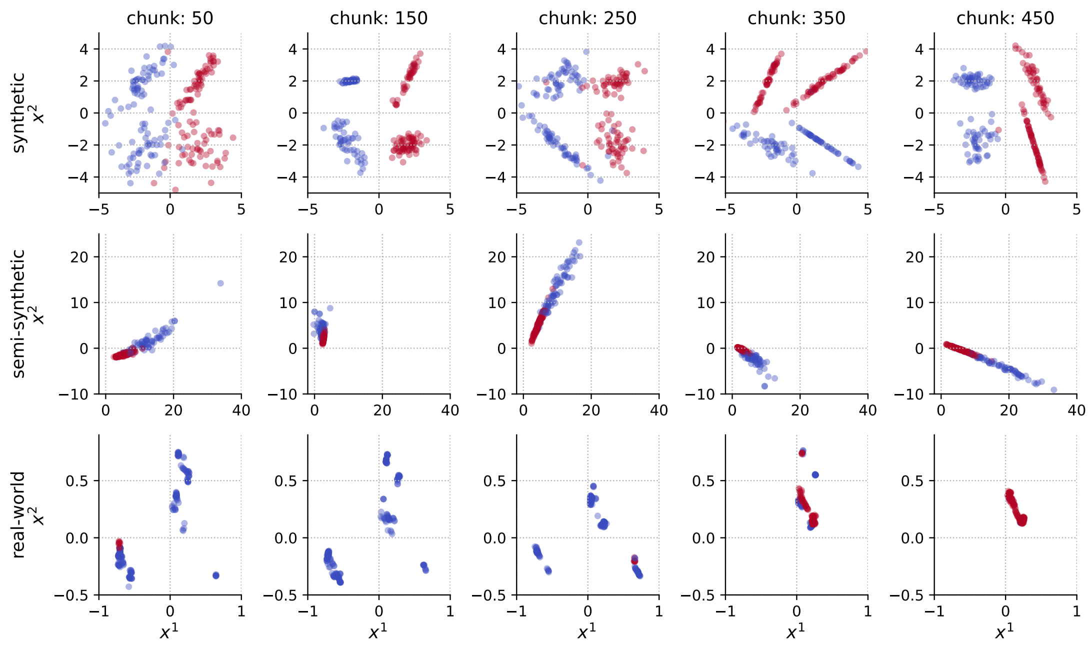

Figure 2.9: *Examples of data streams from each source. The first row presents a synthetic 2-dimensional balanced data stream. The second row presents a semi-synthetic data stream generated based on breast cancer wisconsin dataset with feature projections to two dimensions. The final row shows a real-world covertype data stream, with features transformed to two dimensions using* PCA *fitted to the first data chunk.*

present synthetic, semi-synthetic, and real-world data streams, respectively. The synthetic data stream was generated using *stream-learn* library [133], semi-synthetic modified a *breast cancer wisconsin* dataset with random feature projections [117], and the real-world example presents a *covertype* dataset that has been binarized and transformed to two-dimensional feature space using PCA. Each stream consisted of 500 chunks, each aggregating 200 data samples.

The synthetic and semi-synthetic data streams presented in the figure are described with five artificially induced concept drifts. Hence, each column presents a different concept. In the real-world example, the concept drift ground truth is not available. However, the differences in the data distribution are clearly visible.

In this dissertation, the largest number of experiments is conducted on synthetic data streams, considering the possibility of their vast adaptation for specific problems considered in the research. The specification of generators' hyperparameters allows, among others, the evaluation of streams with a given number of features, with the indication of their informativeness, as well as testing methods on streams with various frequencies and types of occurring drifts [23]. Some of the presented studies were also conducted for real-world streams. However, the lack of possibility of unambiguous evaluation of methods based on data without the concept drift ground-truth drifts is only illustrative and presents prospects related to the explainability of detection. The studies also partially use semi-synthetic streams, in which instances from static *computer vision* datasets are synthetically sampled in order to simulate the time-varying level of difficulty of the data stream.

2.3 Concept drift detection

The phenomenon of concept drift has already been formally defined in Subsection 1.1.5 and described in detail in Subsection 2.2.2. Subsection 1.1.5 also described the quantitative measures proposed by Webb et al. [230], whose primary motivation was to enable the objective detection of a concept change.

This section focuses on the critical task of concept drift detection, presenting the state-of-the-art solutions developed for this purpose. The available solutions are divided into the two main taxonomic groups – performing explicit or implicit drift detection. Additionally, the unsupervised methods are described in detail as an important subset of implicit approaches. The final subsections present the aspects critical to the drift detection task evaluation, describing the metaestimator necessary for explicit drift detection and the strategies of detection quality assessment.

Concept drift detectors are modules that aim to minimize the adverse effects of concept change. They incrementally monitor the data distribution to present a binary information (or sometimes a three-state information [70]) about the occurrence of a drift or its absence.

Concept drift detectors may operate as an independent module, or since concept changes may directly impact the recognition quality, they may be integrated with other methods as a part of a *hybrid* system [39, 128, 189]. Their integration with classifiers allows detectors to directly monitor the error rate of a system, which may reveal the *real* concept drift. Additionally, such integration allows the recovery of recognition quality after the drift occurrence – where the usual procedure used in the case of detection is to replace the classifier with a new one [17].

2.3.1 Methods

Drift detectors can be divided into two main families: (a) *explicit* drift detectors, monitoring errors made by the classifier and detecting drift directly based on the baseline classification quality, and (b) *implicit* ones, detecting a concept change based on other data properties. The specific category of implicit drift detectors are the *unsupervised* ones, recognizing changes in the scenario of label absence, common in the case of high velocity of data streams.

Figure 2.10 presents the taxonomy of drift detection methods, dividing them first into *explicit* and *implicit* approaches and then into *supervised* and *unsupervised*. The diagram presents as well three categories of monitored factors used to detect the concept drift: (a) performance measures, (b) properties of the classifier, and (c) properties of data [114].

All *explicit* drift detection methods require labels to assess the performance of the classification model, which assigns them to the *supervised* category. Some *implicit* approaches monitor the properties of the classification method, while others monitor the properties of data related to the distribution of samples. Both of those factor families can be used in *supervised* or *unsupervised* drift detection methods. Monitoring the classification model during the inference process is possible with or without label access. Some unsupervised approaches may, however, still require training the recognition model [202]. Finally, monitoring data distribution can rely on labeled or unlabeled examples depending on the used measures.

The monitored factors can be viewed as the *metafeatures* of the analyzed data stream. In the case of the simplest, explicit detectors, this factor is directly related to the current quality of the recognition model. Therefore, it is not a direct measurement of changes

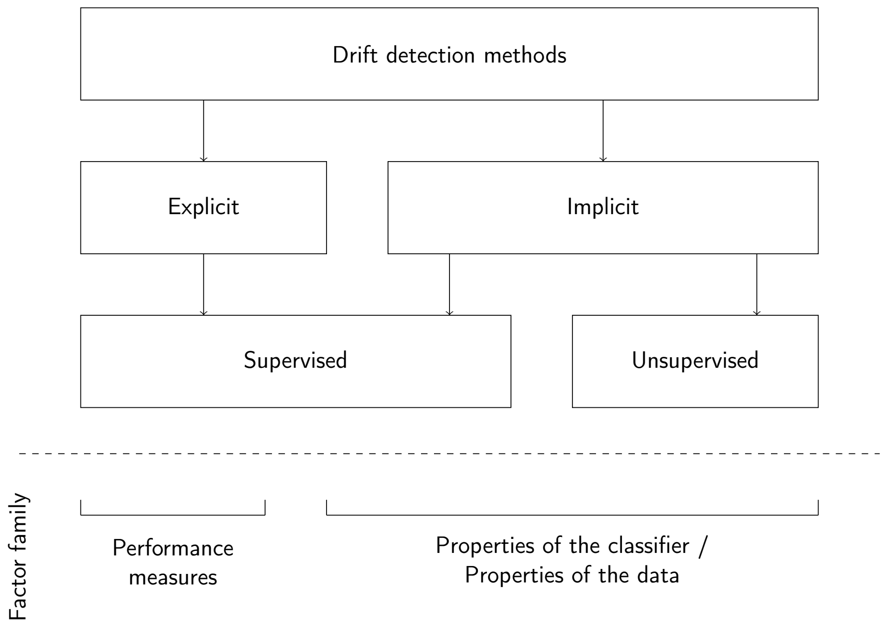

Figure 2.10: *Concept drift detection method's taxonomy and the factor families monitored during the data stream processing. The methods can be divided on two levels: (*a*) into explicit and implicit ones and (*b*) into supervised and unsupervised ones. The drift detection methods use three main factor families.*

in the concept but an indirect verification of their impact on the model, measured with the necessity of class labels. In the case of implicit drift detectors, those *metafeatures* rely on properties other than the classification quality of the underlying recognition method. So far, most of the solutions available in the literature select a single implicit metafeature or a small group of them depending on a given processing context [40, 87].

The research presented in this dissertation focuses on the methods that utilize a range of metafeatures for concept drift detection, showing that a combination of various dataset characteristics can effectively describe the currently processed concept – allowing for *implicit* concept drift detection.

Explicit drift detection *Explicit* or *performance-based* methods rely on monitoring the recognition quality of the classification method. They utilize the fact that *real* concept drifts affect the decision boundaries of the problem, and therefore, the classifier's ability to recognize samples from the new concept is reduced. Some of the first and still frequently used drift detection methods belong to this category. According to the definition by Kuncheva [135], explicit methods directly act upon the drift in case of detection. Hence, there is a particularly close connection between the drift detection method and the classifier on the implementation and conceptual level.

Figure 2.11 shows the general scheme of the explicit detector's operation. With a stable concept visible at the beginning of the processing time, the classification quality remains similar. As a result of changes occurring in the data, the classification quality gradually decreases. An explicit detection method operating in a three-stage mode first signals a warning, and if – as in the presented example – the quality continues to deteriorate, it signals a drift. Explicit methods are usually based on a mechanism that compares the distributions of results obtained by the classifier which is combined with the detection method.

The close relationship between the detector and the classifier typical of explicit solutions allows for an almost immediate reaction in the form of a reconstruction of the classification method when a drift occurs. Depending on the implemented approach, some methods may start preparing to replace the old classifier with a new one already at the warning stage [56]. The dotted black line visible in the figure shows how the classifier would perform in the case of no detection or the case of no reaction by the recognition method. The classification quality remaining at a low level in such a scenario emphasizes the importance of monitoring and reacting to concept drifts.

As mentioned, the precise mechanism of explicit drift detection depends on the method. The most direct approach was used in *Drift Detection Method* (DDM) [70] and *Early Drift Detection Method* (EDDM) [12]. The first solution detects drifts based on the error probability of the underlying classifier. It utilizes the assumption that the classification quality should increase or remain constant with the stream processing time. If otherwise, the method detects a change of concept. A similar strategy was used in EDDM, where the distances between errors are monitored, assuming that these distances should increase

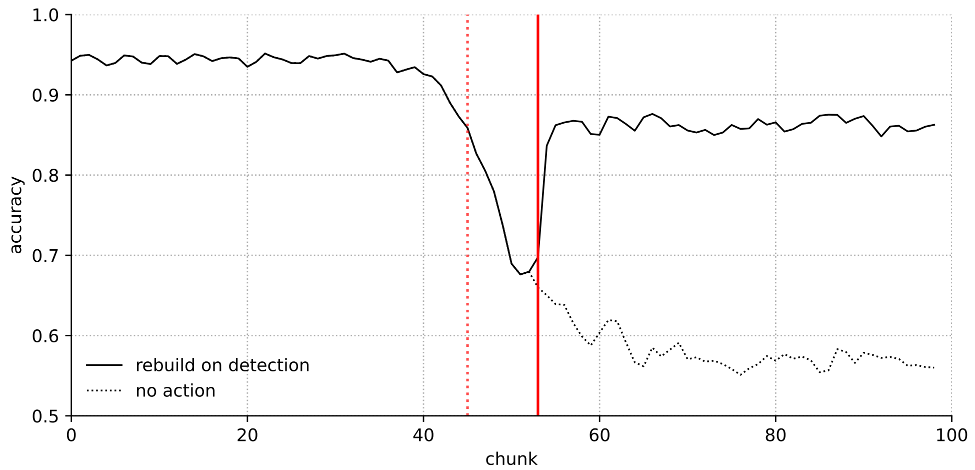

Figure 2.11: *The scheme showing the moments of drift warning (vertical dotted line) and drift detection (vertical solid line) of explicit drift detection method. After the detection, the classification model can be rebuilt to adapt to the changing data distribution. When no action is performed, the classification accuracy continues to decrease, as shown by the dotted black line.*

with processing time. As in the case of DDM — if distances between errors decrease — a concept change is signaled.

A more complex approach to drift detection was used in *Adaptive Windowing* (ADWIN) [24] – a detector using sliding windows of varying width. The principle of sliding windows operation is based on comparing specific values for two windows — one describing historical data, the other – recent data. Drift is signaled when a difference between the two windows is observed. This drift detector finds many applications in modern ensemble approaches [25, 26, 39].

The windowing approach was also used in the *Paired Learners* (PL) [11] method. The PL algorithm uses two recognition models — the static one, trained with all incoming patterns, and the reactive one, trained only with patterns from the recent past. If the classification quality of the static model falls below the classification quality of the reactive classifier, a concept change is marked.

The performance-based detectors also include methods that analyze the statistical properties of errors made by the classifier, such as the *Hoeffding Drift Detection Method* (HDDM) [66], which uses Hoeffding inequality to detect significant changes in the sliding average classification quality. The method can use two strategies: (*a*) moving averages (HDDM$_A$), which is more suitable for sudden changes in concept, and (*b*) weighted moving averages (HDDM$_W$), dedicated to gradual concept drift detection.

Some *ensemble* approaches to drift detection have also been proposed, utilizing more than one mechanism for improved detection quality [48]. A simple and lightweight solution was presented in *Drift Detection Ensemble* (DDE) [159], which integrates three explicit drift detection methods, selecting the set of base methods depending on the sensitivity specified by the hyperparameter. A similar approach was used on *e-Detector* [56], which, however, does not limit the number of ensemble members to three and additionally performs clustering to select a single method from the similarly performing group. The assumption of *e-Detector* is to select the best-performing approach for a given data stream using a static selection. Both mentioned ensemble solutions utilize the output from the shared classifier to monitor the errors made by the classifier. This strategy allows to limit the computational complexity of the ensemble method. On the other hand, the members differ only on the level of the detection mechanism, which significantly limits the *diversity* of a detector's pool.

The particular drawback of explicit drift detectors is their classifier dependence. Such methods do not allow for the detection of virtual drifts, initial phases of real drift (which do not yet harm classification quality), and some changes in prior probability, especially when the errors are measured on a general level instead of a class-specific level [215].

Moreover, the operation of explicit drift detectors strongly depends on the classifier used in the processing task. Using different baseline algorithms may result in different detections, as the *bias* of the classifier is dependent on the algorithm and – in the case of non-deterministic learners – also on the initial state of the model.

Implicit drift detection Implicit methods, sometimes denoted as distribution-based, monitor factors other than the classification quality of the underlying recognition method to detect concept changes [100]. This group includes supervised methods that analyze the statistical properties of the data, taking into account the sample's labels [85] as well as unsupervised methods, with a significant predominance of the second category. Therefore, according to the factor family taxonomy, as presented in Figure 2.10, two types of factors are monitored: (a) those dependent on the classification model and (b) those dependent on the data distribution itself.

Among the implicit drift detectors, the *Centroid Distance Drift Detector* (CDDD) [113] method can be mentioned. The approach analyses the distance between the centroids of subsequent batches, directly utilizing the metafeatures of the data batches. This method can operate in supervised or unsupervised mode, depending on whether the centroids are calculated for separate categories of objects or the entire data regardless of their label.

A different supervised strategy was proposed in the *Statistical Drift Detection Method* (SDDM) [163]. This detector provides, in addition to the detection itself, information about the source of the drift and its nature. The method uses a series of functions to determine the drift characteristics based on *concept drift magnitude* [231] – measures proposed by Webb et al., described in detail in Chapter 1, that aim to assess the distance between two concepts based on their probability distribution. The work introducing the method showed that such measures become uninformative in feature spaces of high dimensionality.

Among the methods based on the properties of classifiers, one should mention the *Margin Density Drift Detection* (MD3) [202], in which the density of samples near the decision boundary of the SVM classifier is examined. Similarly, in the *Confidence Distribution Batch Detection* (CDBD) [150] algorithm, the confidence of a classifier is monitored. Meanwhile, the *ExStream* algorithm [50] monitors yet another classifier-related factor – the variability of its explainability.

All mentioned approaches that monitor the classification model, despite their unsupervised detection, require access to labels in order to rebuild the monitored classifier in the initial phase of the processing and in the case of recognizing a concept drift, to further carry out the detection task.

Unsupervised drift detection The particular disadvantage of supervised methods, both from the explicit and implicit categories, is the requirement of almost immediate access to labels. Regardless of how labels are used (whether to monitor the error rate of the classifier or to calculate specific metafeatures of data), the assumption of their almost immediate availability is not realistic due to their general limited access [202], their cost [152], and the possible time delay [86].

Some algorithms use statistical tests, which do not examine the distribution of classification errors (performed, for example, by HDDM) but only the statistical distribution of available data. For instance, detectors may use the Kolmogorov-Smirnov test [210], Kernel Density estimation [211], and correlation of feature distribution in data windows [142] to detect changes in the distribution of unlabeled data.

The data distribution is monitored in the *Nearest Neighbor-based Density Variation Identification* (NN-DVI) [151] detector using the *k-Nearest Neighbors* algorithm. Similarly, a grid-based approach was proposed in the *Grid Density-based Clustering* (GC3) [203]. Other methods study the variability of the data density, for example, using clustering algorithms. The *OnLIne Novelty and Drift Detection Algorithm* (OLINDDA) [213] uses the *k-means* algorithm. Concept drift is signaled if newly incoming data samples do not belong to predetermined clusters.

There are also methods based on the analysis of outlier observations, such as *Fast and Accurate Anomaly Detection* (FAAD) [146], proposed mainly for the purpose of anomaly detection. In *One-Class Drift Detector* (OCDD) [85], a one-class classifier is used to examine the percentage of objects recognized as not belonging to the recognized concept. A similar strategy was used in *Discriminative Drift Detector* (D3) [84], where a discriminative classifier is used instead of a one-class classifier to explicitly recognize objects of the new concept from those from the previous one in variable-width windows.

Despite the benefits of using unsupervised detectors, it should be remembered that these methods do not allow to detect changes in which the drift occurs only in the *conditional probability* of labels $P(\mathcal{X}|\mathcal{Y})$ without affecting the underlying *covariate probability* of the data $P(\mathcal{X})$.

2.3.2 Metaestimator for concept drift detection

Since *explicit* drift detectors operate based on errors made by the classification method, it is necessary to integrate the detection method with the underlying classification algorithm. While in the *implicit* approaches, the classifier is not a necessary component to detect the changes, the detection can be directly used to rebuild the classification model to counteract the loss of recognition ability regardless of the mechanism used to detect the concept changes.

A metaestimator is a simple method combining the work of classification and detection modules, allowing for the state exchange between the classifier and the drift detector. On the implementation level, the metaestimator consists of two components: (a) classifier and (b) concept drift detector. The detection method uses the classifier responses to analyze its errors and to perform drift detection. When a concept change is detected, the underlying classifier is rebuilt to prevent degradation of classification quality. The operation principle of the metaestimator is presented in Figure 2.12.

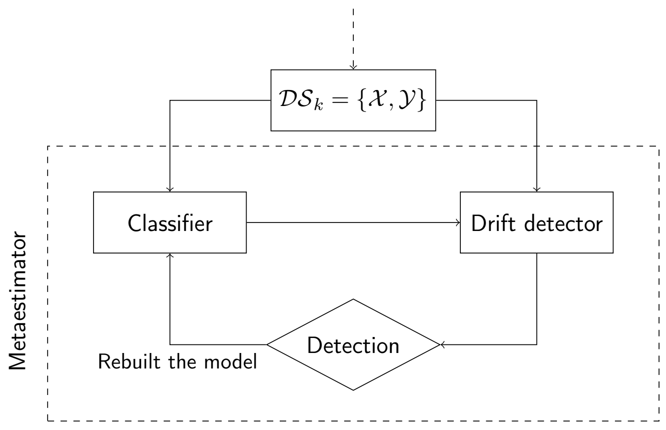

Figure 2.12: *The scheme presenting the operation strategy of metaestimator. The method combines the classification and drift detection modules. After the drift detection, which is performed according to the predicted and real labels of samples, the classifier can be rebuilt, allowing the adaptation to a new concept.*

For each portion of data containing labels, the first step is to perform a prediction using the classifier. The labeled samples are transferred with the actual labels to the detection method, which compares the classifier's predictions with the ground-truth and, based on various criteria embedded in the detection method, decides whether the drift occurred. In the absence of detection, no actions are taken. On the contrary, in the case of drift detection, a complete reconstruction of the classifier is enforced, which allows for the adaptation to the currently processed concept and provides the forgetting of the previous knowledge.

Considering the fact that, in most cases, the real concept drift negatively impacts the recognition quality, this approach could potentially allow for the evaluation of drift detectors based on the classification quality. However, such strategy performs the detection assessment indirectly, which makes it sensitive to many other factors.

2.3.3 Assessment of drift detection task

Measuring the effectiveness of the drift detection methods with the classification quality during stream processing is derived from the most common and straightforward type of action performed in the case of the concept change, aimed at maintaining the quality of the classification – retraining the classifier or replacing the current one with a new instance of the trained model [158]. This typical reaction to a concept change is also embedded in the metaestimator presented in Subsection 2.3.2.

However, available works [22] sensibly advise not to evaluate drift detection methods using the overall classification quality. The research has shown that the reference method that does not detect changes but artificially simulates the detection – by signaling the drift every fixed number of instances – can achieve better results than all of the examined *state-of-the-art* drift detectors.

These observations were confirmed in the other research [126], where the detector signaling the drift in each data chunk achieved the best results of the average classification quality. Therefore, evaluating the detection quality based on the stream's average classification quality rewards *hyperactive* detectors, making frequent and redundant detections desirable. It should be emphasized that re-initializing the classifier or retraining it may be a computationally expensive procedure. Valuable detectors should, therefore, have a low number of false, redundant detections and accurately recognize the exact moment of change. As a result of such observation, the work states that other indicators, independent of the classification accuracy, are needed to assess the detection quality. The mentioned publication proposes three basic measures for assessing drift detectors: *Mean Time between False Alarms*, *Mean Time to Detection*, *Missed Detection Rate*, and two aggregated measures: *Average Run Length* and *Mean Time Ratio*.

Before the drift occurs, all detections are treated as a false alarm, and after the drift – the first detection is considered a true alarm. Those assumptions may raise some concerns in the case of gradual and incremental drifts, where the unequivocal moment of the concept change is indistinct. Additionally, according to the mentioned publication, the ideal detector's average time between false alarms should be high. In the case of several detections signaling one non-sudden drift of long duration, the value of this measure is low. However, one can consider such behavior of the detector as desirable, as adapting the classifier to a temporal concept can improve the operation of a system during the changes of low dynamics.

Drift Detection Errors Since the metrics available in the literature are not well-suited for evaluating non-sudden concept changes, a new set of measures was designed for the reliable experimental evaluation of a broad range of existing concept drift dynamics.

In the experiments conducted for this dissertation, the drift detection quality is assessed according to three novel measures analyzing the distances between the detection and the actual drift, as well as the number of signaled concept drifts. Those measures are better suited for gradual and incremental changes, where, according to the method's sensitivity, the drift can be signaled in the initial or later stage of concept transition. Moreover, the drift with low dynamics can be signaled in many stages, marking the transition phase as an independent concept.

The proposed *drift detection error* measures, designed to evaluate concept drift detectors regardless of the dynamics of changes, assess detectors based on three fundamental, non-aggregated criteria, offering an extended interpretability of the method's operation. While the integrated measures may be easier to interpret, they will provide less information about the quality of considered task, compared to the individual analysis of valuable criteria [232].

The *drift detection error* measures are defined as follows:

$D1$ – *The average distance from each detection to the nearest drift*, described as:

$$D1 = \frac{\sum_{k=1}^{n_{r'}} |r_c - r'_k|}{n_{r'}}, \tag{2.5}$$

where $n_{r'}$ is the number of signaled detections, r' is the set of detection moments, r is the set of drift moments. Value $|r_c - r'_k|$ describes the distance from detection r'_k to the closes drift r_c. Hence, c is the index of the nearest actual drift occurring in the stream.

This measure penalizes the methods with numerous redundant drift detections, occurring far from the moments of actual concept changes. The average distance from detection to drift increases in the case of hypersensitive methods. However, this measure does not consider the unrecognized concept changes – hence, methods of low sensitivity are not penalized for lack of specific drift detection.

$D2$ – *The average distance of each drift to the nearest detection*, described as:

$$D2 = \frac{\sum_{k=1}^{n_r} |r'_c - r_k|}{n_r}, \tag{2.6}$$

where n_r is the number of drifts occurring in the data stream and the value $|r'_c - r_k|$ describes the distance from the actual drift r_k to the closest detection r'_c.

This measure considers the closest detection of each actually occurring drift. In this measure, in contrast to $D1$, the methods are penalized for lack of drift signalization. However, the hypersensitivity of drift detectors has no impact on this error measure – as only the single detection closest to drift is considered.

R – *The adjusted ratio of the number of drifts to the number of detections*, described as:

$$R = \left| \frac{n_r}{n_{r'}} - 1 \right|, \tag{2.7}$$

where $n_{r'}$ is the number of detections signaled by the method and n_r is the number of drifts occurring in the data stream.

This error measure considers the number of drifts that actually occur in the stream and the number of detections signaled by the method. The lowest possible error value of zero is achieved for a method signaling the exact number of changes as a number of drifts. If the method signals multiple redundant detections, the error rises towards the value of one. If the method signals fewer detections due to low sensitivity, the error can rise to values exceeding one. This measure, therefore, penalizes both too many and too few detections. Nevertheless, too few detections raise the error value, assuming that the lack of drift recognition is of greater significance when processing the non-stationary data stream.

It is essential to note that measures can only be defined if the evaluated method signals any detection. By definition, the errors for a scenario of no detections are infinite. In the presented research, to enable the comparison with methods that did not signal any concept changes, the case of no detections is treated equivalently to the case of signaling concept drift in each of the processed chunks. This approach to evaluation has been motivated by the supposition that a method not signaling drift carries as little information about the data as a method signaling drift in each data chunk.

Both sets of measures, proposed by Bifet [22] and the presented *drift detection error* measures, require access to concept drift ground truth, describing the moments of actual concept changes occurring in the data stream. As described in Subsection 2.2.3, this restriction limits the possibility of the concept drift evaluation for real-world data streams. The lack of unequivocal markings of types and moments of drifts makes experimental evaluation on this type of data complex and highly inconclusive [158]. Meanwhile, the synthetic and semi-synthetic data streams allow for easy access to the moments and types of drifts.

As the presented measures evaluate the distances between detections and concept drifts, a single marking for each concept change is needed. The sudden changes offer such an opportunity by default – the ground truth indicates the point after which the following data samples belong to a new concept. In the case of incremental and gradual changes, however, the drift is stretched over many samples and even for many data chunks. In this scenario, it was decided to mark the concept drift point at the central moment of the concept transition period. Any detection not directly overlapping with the central moment of drift causes an increase in the overall drift detection errors based on distances ($D1$ and $D2$ measures), even though the concept change is in progress. The multiple markings of the exact concept change – for example, the recognition of temporal concept by marking detection in the initial and final moments of concept transition – on the other hand, influences the error responsible for the ratio of drifts to detections (R).

The mentioned example of non-sudden concept drifts shows that the evaluation of concept drift detection using a quantitative measure is a complex task. The proposed measures for assessing the detection quality stand an imperfect but valuable compromise intended to allow for such assessment regardless of the type of drift occurring in the streams.

Visual assessment of drift detection The quantitative quality assessment using *drift detection error* measures can be supplemented with the visual assessment of the drift detection, aiming to provide a better understanding of the method's operation – by analyzing the moments of detection in relation to actual drifts – over the entire stream course and the performed replications.

Such a visual assessment does not provide an objective measurement of detection quality. However, as an addition to the already presented quantitative measures, it may allow for noticing the flaws of some methods and – in particular – show moments when the assessed method is hyperactive or fails to recognize the concept change correctly.

Figure 2.13 presents the scheme showing the visual assessment methodology. The plot shows the detection of an abstract method over ten replications. The consecutive chunks are presented on the horizontal axis of the plot, with the moments of actual concept changes (known from the stream ground-truth) marked with vertical dashed lines. The example shows the fragment of a data stream with four concept drifts. In such a scheme, the detections of a method are marked with circles stacked vertically following

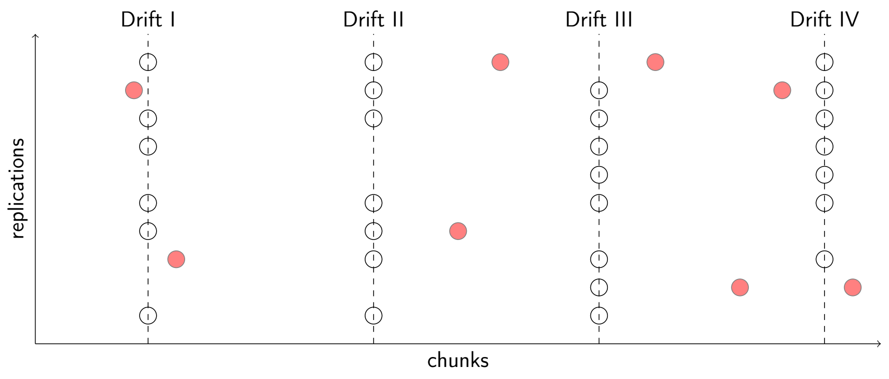

Figure 2.13: *The scheme showing the adopted way of the drift detection quality visual assessment. The strategy is built upon comparing the moments of drift and the moments of detections across the experiment replications.*

the experiment replications. For a better distinction, the redundant or late detections are marked with red color. The ideal method's performance would be indicated by the circles overlapping with the vertical dashed lines across all replications.

As stated in the description of the *drift detection error* measures, the evaluation of gradual and incremental concept changes needs a more complex interpretation. The vertical line indicates the central moment of drift transition, and – depending on the sensitivity of the method – the concept change can be recognized before or after the drift equilibrium point.

The visual interpretation of detection moments in relation to the concept drift ground truth and the possibility of assessing the method's sensitivity to gradual and incremental changes is of essential value for a reliable and thorough evaluation of the method. This motivates the use of such an analysis in the presented research, in addition to a quantitative and direct evaluation using *drift detection error* measures.

2.4 Data stream classification

Subsection 1.1.2 formally defined the classification task and Subsection 1.1.4 – the data streams as a specific data structure. As mentioned, among the requirements of effective data stream processing, the continuous adaptation of the model is necessary. The classifier ensemble was presented as an exemplary solution to such a requirement.

This subsection describes methods widely used for data stream classification. The available solutions were divided into two main groups: the built-in mechanisms and the hybrid mechanisms. The second group follows the ensemble approach to classification, which is discussed in more detail in this section. Additionally, the section *has been supplemented with a description of classification quality metrics and processing protocols for data stream classification, allowing for the reliable evaluation of the methods.*

As previously noted, classification remains one of the most frequently performed machine learning tasks, exhibited by various solutions proposed over the years of artificial intelligence development [164]. It becomes natural that considering the classification task in a data stream setting, which enforces specific requirements on the processing methods [55], does increase the number of possible approaches due to (a) the variety of adaptation techniques to the difficulties coming from processing of this type of data, and (b) to a range of possible scenarios related to time variability of the data, that affect the classification quality. Those include concept drifts, including the changes in prior probability, the possible emergence of new classes, and varying costs of feature acquisition.

There are two main categories of classification approaches used in data stream processing: (a) utilizing the *built-in* mechanisms of adaptation and (b) *hybrid* approaches, using the *classifier ensembles* to allow the model's adaptation to incremental learning and time-evolving data streams [132].

2.4.1 Built-in adaptation mechanisms

The first group of solutions have a natural ability of incremental learning or have been modified so that a single model can accumulate knowledge over a processing time. The key in this context is the ability of the classifier to forget the outdated knowledge and synthesize the current one by incremental processing of the new instances [185]. Among those approaches, there is a range of *state-of-the-art* classifiers, sometimes extended by sophisticated adaptations to allow for their incremental learning.

Gaussian Naive Bayes (GNB) classifier is a method that has a natural ability to process samples incrementally. This method updates the class-conditional probabilities throughout the processing [134]. The modeling of class probabilities embedded into GNB classifier offers a great benefit when processing data of infinite volume – as the number of processed samples does not influence the number of parameters stored in the model. However, by default, each data batch used to fit probability distribution is of equal weight. This means that in the case of a concept drift, the historical data distribution influences the decision for the currently processed concept proportionately to the number of samples from a given concept used to fit the model [140]. To solve this problem, Krawczyk and Woźniak proposed a weighted version of a method [129], where the level of importance of old samples is decreasing, to the point of discarding their impact and removing old samples from the model memory.

k-Nearest Neighbors Classifier (KNN) is another commonly used approach, valued for its intuitiveness and accuracy. Its operation principle in a canonical form is to preserve the original training set in the model's memory. During inference for a given test sample, labels belonging to the k nearest instances of the training set are taken into account [139]. However, such a principle of operation does not work well in the case of incremental learning due to limited memory resources. It should be recalled that according to the data stream processing guidelines presented by Domingos, presented in Subsection 2.2.1, it is forbidden to allow the accumulation of processed samples in the memory due to the need to maintain a constant and short inference time for each sample [55].

In adaptation to that constraint, the extensions of the *k-Nearest Neighbors* algorithm were proposed, which are more suitable for incremental learning. The approaches first modified the set of samples stored in the model memory [91]. Later, Aha et al. [6] proposed an instance-based learning framework. Those extensions achieved satisfactory recognition results, allowing for a significant reduction in the memory needed to store samples.

Decision Tree Classifier (DT) is a popular classification algorithm [187], especially appreciated for its simplicity and explainability [245]. Among the most popular implementations of decision trees one can name the C4.5 algorithm [188]. It uses the greedy approach to maximize information gain or gain ratio when building the tree structure and then later prunes it with a single-pass algorithm to avoid overfitting. This canonical implementation of a decision tree requires recursive scanning of the available dataset to determine the tree structure and splitting criteria [103, 175].

To adapt a decision tree to incremental learning, the *Hoeffding Tree Classifier*, embedded in the *Very Fast Decision Tree* (VFDT) algorithm, was proposed [54]. This algorithm uses the Hoeffding inequality to calculate the sample size needed to determine the splitting criterion, which allows building the tree structure after a single pass of the available data, which is crucial for systems dedicated to data streams [55]. However, this algorithm was not adapted to a varying concept, as the tree nodes could not be modified after their creation. Therefore, an extension of the algorithm in the form of a *Concept-adapting Very Fast Decision Tree* (CVFDT) [103] was proposed, allowing for detection and adaptation to changes in the concept by using sliding windows. When the optimal division criterion in the existing branch of a tree changes, an alternative sub-tree is created, which classifies instances from the new concept. Outdated sub-trees are later being removed.

Artificial Neural Networks (ANN) is yet another classifier with natural forgetting abilities [27, 168]. As already briefly described in Chapter 1, such a model consists of a layered arrangement of individual units, denoted as neurons. Each unit has a specific weight, and their connections have a specific bias. As a result of sample propagation through such a system of neurons, the network output presents weights corresponding to the probability of class membership. During classical network training, as a result of many *propagation* and *backpropagation* sequences, the optimal weights and biases are sought.

In the case of data streams, the traditional approach of epoch-based neural network training – where the entire training dataset is divided into batches and used multiple times to optimize the model's weights – is abandoned in favor of using individual samples (in online processing) or data chunks (in batch processing) of the data stream [134]. During continuous training, a natural adaptation of the algorithm to the current concept occurs in the form of weight modification based on the current learning rate [71]. This default strategy of weight optimization indicates that the newest data describing the current concept, once used to fit the model, has the most significant impact on its parameters.

Such behavior of neural networks is described as catastrophic interference phenomenon [64]. It causes the model to *forget* of tasks effectively learned in the past once updated with new knowledge. Once assuming that the concepts visible in the data stream are non-recurring and the complete knowledge about the processed task is available in the currently processed data, the catastrophic inference becomes a desired feature of neural networks, allowing for an automatic adaptation. However, in some incremental learning scenarios [224], where the objective is to learn new tasks while maintaining the ability to solve the previously mastered ones, the catastrophic inference becomes an impediment.

2.4.2 The fundaments of ensemble learning

While the data stream processing is the main objective considered in this dissertation, the classifier ensembles are not limited to this learning environment. Such solutions, also denoted as *Multiple Classifier Systems* [10, 241], are also commonly used for static data [1, 179], offering improvements in the classification quality and a possibility of combining simple models to solve complex tasks [46]. The particular benefit of combining decisions made by a group of independent voters is described by *Condorcet Jury Theorem* – saying that if each voter has an above 50% probability of making a correct decision in a binary problem, the combined decision is liminal to 100% with an increasing number of voters [205]. Such strength of combined decisions is intuitive to humans, having their origins in the *wisdom of crowds*, prominent for centuries in judiciary and law [99], and nowadays also utilized to annotate the large quantities of data by crowdsourcing [233].

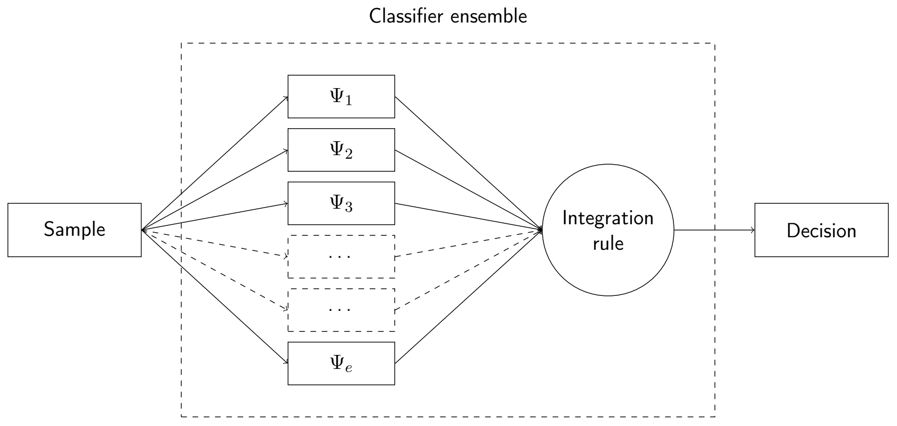

Figure 2.14: *A scheme of an exemplary ensemble classifier with a parallel topology. The data sample is provided to each of the models in the pool, and their predictions are integrated by a* fuser *to form a final decision.*

In the context of the classification task, the *voters* or *experts* are the classifier instances combined into an ensemble. The scheme of such a system is presented in Figure 2.14.

In the canonical ensemble system, each classifier Ψ_k in a pool $\Pi = \{\Psi_1, \Psi_2, \dots, \Psi_e\}$ makes the decision for each input sample. The individual decisions are integrated using an integration rule specific to a designed system [240]. It is worth mentioning here that classifier ensembles can have complex topologies – of which *parallel topology*, presented in the figure, is the most intuitive and common [244] – as well as various strategies of input data modifications and a wide range of decision integration mechanisms.

From the machine learning perspective, the classifier ensembles can be described as *hybrid* systems – meaning, they combine different mechanisms to improve the overall recognition quality. The word *hybrid* originates from biology but has been adapted to a general context, where it describes a mixture of multiple different components [206]. The use of a classifier ensemble offers, therefore, a possibility of hybridization of multiple individual systems, each having its strengths, so that after the integration of their decision, the overall classification quality outperforms the individual components of a system [179, 240].

One of the first works using such a hybrid classifier was presented by Dasarathy and Sheela [46]. The authors combined linear classifier with *k-Nearest Neighbors* and showed that the hybrid system – selecting the appropriate component depending on the region of the feature space – offers better classification quality than the recognition using individual, baseline classifiers used in the system, as well as limits its computational cost.

The classifier designed by Dasarathy and Sheela used a static model selection – the area of the feature space directly indicated the selected classifier. As mentioned, the modern approaches can utilize various mechanisms, not only on the level of decision integration but also when constructing the ensemble. The following paragraphs describe the most critical processing stages of currently used methods: (a) ensuring the ensemble's diversity, (b) integration of decision, and (c) pruning of a classifier pool. As those stages are fundamental for both static and streaming data processing, the aspects in the context of static data are first described, followed by data-stream-specific comments.

Ensuring Diversity To use the benefits of *Condorcet Jury Theorem* in classifier ensembles, the *independence* of individual models is required. Such independence means that the output of an agent is not influenced by other agents around [193] and manifests itself in high *diversity* of decisions.

One can imagine a system where all classifiers return an identical verdict for a given sample. In such a case, regardless of the integration method, the output of the hybrid system is equivalent to a label returned by any embedded model [80]. The only consequence of using such an ensemble is an increase in the system's computational complexity. Therefore, the system designers need to ensure that each classifier in a pool has its region of competence. When the classifiers return correct decisions in their competence area, and outside – a random decision – an appropriate integration rule and correctly distributed areas of competence allow the use of Condorcet's proofs and increase the classification quality relative to the base systems.

Several measures have been proposed to assess the diversity of a classifier ensemble [138]. Those measures analyze the outputs of pool members for given samples. The lowest possible diversity is typical for an extreme case where all classifiers return identical output for the given sample.

Ensuring the diversity of ensemble members is performed at the stage of classifier training. There are two main strategies on the most general level – depending on the components of a hybrid system, the pool can be:

- *Homogenous* – when the pool consists of the models working according to the same algorithm, and, therefore, the diversity must be assured by modifying the original dataset. If the exact models were trained using the same data, their outputs would be identical for any sample – in the case of a deterministic learner – or very similar – otherwise.
- *Heterogenous* – when the pool comprises various *algorithms* or their different versions. The first classifier ensemble designed by Dasarathy and Sheela was a heterogeneous system consisting of a linear classifier and KNN. Similarly, a heterogenous ensemble system was presented in Subsection 1.1.5. The described *Drift Detection Ensemble* [159] used a pool of three different drift detection methods, selected

depending on the defined sensitivity of the method. This example prompts that ensemble methods are not limited to classification tasks.

Combining various classification algorithms already offers diversity due to their internal bias [179], so additional modifications on the dataset level may not be needed. In the case of using the same base learner, dataset manipulation is necessary to enforce diversity. There are two main categories of the data manipulation techniques:

- *Training classifiers with different data partitions.* Solutions include *bagging* and *boosting*. This approach is presented in the top right of Figure 2.15.

  Bagging generates several subsets of data using sampling with replacement. The samples are randomly drawn from the original dataset and are used to train a single classifier. The procedure is repeated for the other classifiers in the pool, generating a different set of training samples.

  In contrast, *boosting* generates several subsets iteratively. In this technique, the sample that was incorrectly classified in the previous iteration has a higher probability of being selected. In such a way of partition generation, the most attention is given to *complex* samples – possibly improving the classification quality in the difficult feature space region, e.g., near the decision boundary. Boosting is a technique suitable for iterative training – when the samples are chosen from the original dataset, similar to bagging – but their selection is not entirely random. The most popular method from this category is *AdaBoost* [65] algorithm.

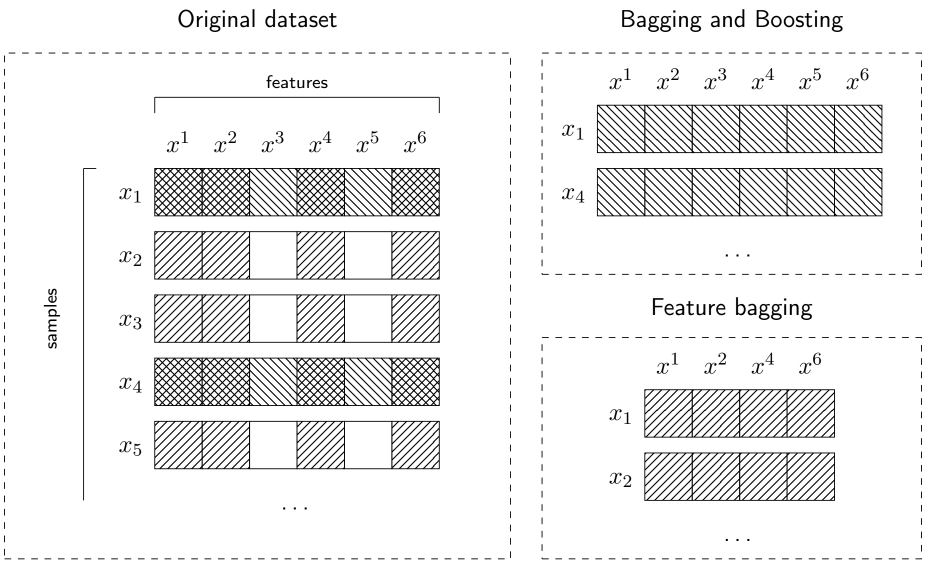

Figure 2.15: *Data manipulation techniques used in ensemble learning. The original dataset shown on the left side is modified following the bagging and boosting strategy – shown on the top of the right side of the figure – and the feature bagging – shown at the bottom of the right side.*

- *Training classifiers with different subsets of features.* One of the techniques is *feature bagging*, sometimes denoted as *random subspace*. Such a solution is presented in the bottom right of Figure 2.15. In the presented example, the single classifier is fitted using four randomly selected features from the original dataset. Analogously to bagging, the following classifiers in an ensemble are fitted with different randomly selected features. This approach is used in a *Random Forest* classifier [32] – a popular algorithm that uses an ensemble of *Decision Tree* models.

Alternatively, the ensemble can follow a *Random Patches* strategy of data manipulation [156]. This technique combines traditional *bagging* with *feature bagging*, allowing for a generation of diverse data partitions used to train the classifiers in a pool. In this approach, not only the data samples are randomly selected, but also, the random subspace is selected for each subset of samples. The lower memory needs of this approach make it suitable for big data and data stream processing [81].

The ensemble's diversity is equally important in the context of data streams. Similarly to static data, heterogenous [225] or homogenous [243] ensembles are proposed, with the vast majority of solutions from the second category [128], which highlights the necessity of data manipulation techniques at the level of classifier training. Additionally, which is a critical problem for data streams, the entire training set never arrives. This indicates that baseline solutions manipulating the dataset must be adapted to an environment with infinitely incoming samples [71]. It is worth pointing out here that storing the arriving samples for future use is not aligned with the guidelines of data stream processing [55]. To solve this issue, Oza et al. [178] proposed online bagging and boosting algorithms.

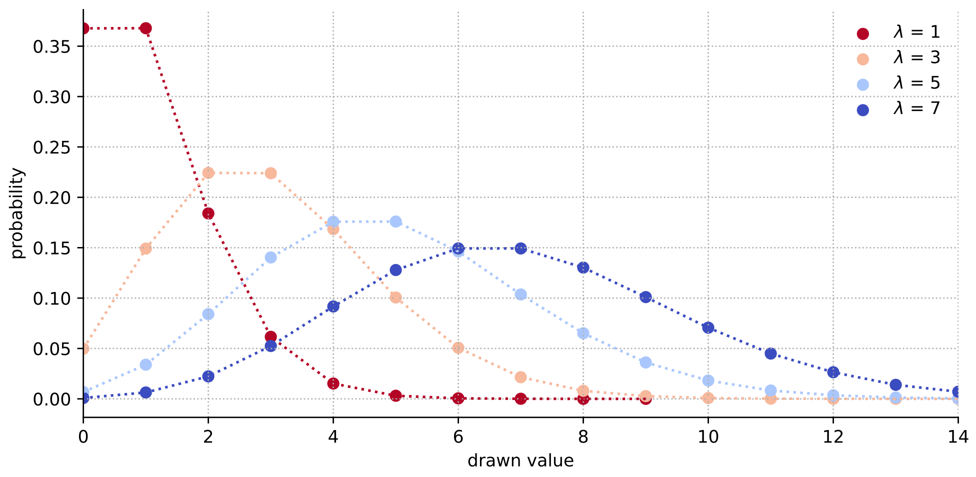

Figure 2.16: *The probability of drawing random values from Poisson(λ) distribution, depending on the λ parameter. The probability was calculated after drawing 100,000 random values from each distribution.*

In the proposed solutions, the authors considered the potentially infinite inflow of samples in the stream and the lack of possibility to accumulate incoming samples for the need of selection with replacement, used in traditional bagging [31]. The proposed solutions utilize the sampling from *Poisson distribution*, which became a standard in methods dedicated to *online* processing of data streams. The probability of selecting values from $Poisson(\lambda)$ for five different parameter values has been presented in Figure 2.16.

The $Poisson(\lambda)$ allows to randomly sample discrete values, which determine the number of classifiers that are trained using a newly incoming sample. At default, in the online bagging proposed by Oza et al., $Poisson(1)$ distribution is used to determine k copies of each incoming sample used to train the models. In online boosting, the weights depending on the correct and incorrect responses of individual models for such samples are calculated and used to select the appropriate λ parameter to sample the number of copies from the distribution [178]. The larger the λ, the more classifiers are fitted with the given sample.

Integration At the inference stage, integrating responses from individual classifiers from the ensemble is necessary – it is done using a module often denoted as *fuser* [240]. This module combines the outputs of ensemble members using a weighting mechanism.

The integration might be handled on two primary levels:

- based on classifier's output [244] – the decision of a classifier can be delivered to the integration module in the form of discrete *vote* or a continuous *support function* – describing the probability of a decision and additionally information about the classifier's *certainty* towards specific class.
- based on weighted factors [242] – the weights of the classifier's outputs can have four different dependencies. They can depend on (a) the classifier, (b) the classifier and the class label, (c) the classifier and the feature vector, (d) the classifier, the class label, and the feature vector.

At the simplest and most intuitive approach to ensemble integration, called the *majority voting*, each member returns a discrete decision, and weights of each classifier are equal – hence, the most frequent *label* among the classifier's outputs is selected. The discrete outputs are also considered in *Stacking* [235]. In this approach, the training stage is divided into two phases: first, fitting the individual classifiers in an ensemble, and second, fitting the method responsible for integrating the classifier's discrete decisions. The stacking approach uses a heterogenous ensemble, which additionally motivates the use of discrete outputs, as the combination of support functions from different algorithms may require their normalization [240].

The continuous outputs expressing the classifier's support towards specific classes can be used in more sophisticated solutions [241]. Only combining discrete outputs can allow for outperforming the *oracle* – an abstract ensemble solution in which the correct decision of a single classifier is explicitly enforced [237]. The integration based on continuous support function was used in *Mixture of Experts* [104], where the answer of the most competent models from the classifier pool was combined to provide the final answer using a *gating network*. Some solutions measure the similarity of the classifier support and the *Decision Templates* – describing the most typical structures of the classifier's outputs for each class [137]. Some strategies of *fuser* design, especially when establishing the classifier weights considering feature vectors, use the *trainable* module [238]. A trained weight optimization module can improve the chances of noticing complex traits when weighing the responses of individual classifiers.

Suppose the weights of a fuser are all equal to zero, except for a single classification model – i.e. the weights are defined using the *indicator function*. Such a scenario describes a *classifier selection* – a process in which a reduced number of classifiers is responsible for a final decision [44].

The *selection* is frequently mentioned when describing the training phase, where ensemble methods aim to enforce the classifier's local specialization [240, 241]. The training data can be divided into partitions on the level of samples or feature subspaces to explicitly define the competence region of an individual classifier [250].

After the classifier is fitted, at the point of decision integration, the selection can be *static* or *dynamic*. In the case of *static* selection – the relation between the competence region and a classifier is fixed. In contrast, in the case of *dynamic* selection, the competence of individual ensemble members is estimated for a given input sample in the inference process, and then – based on evaluated model competence – the single classifier is selected. It is worth noting here that some solutions select a subset of classifiers and then, based on their responses, provide an integrated decision [33]. This case may present a classifier selection as an additional stage between generating the classifier's pool and integrating their decisions.

The data stream processing scenario does not impose any specific challenges related to the decision integration phase. While, as mentioned, on the level of model training, the requirement of continuous adaptation was assured during the incremental training of individual classifiers [178], at the stage of decision integration, the adaptation can be enforced as a result of a re-calculation of classifier's weights [38]. This is especially evident when noticing a slow-pace gradual or incremental drift. Assuming that the new data samples are used to incrementally train available models, at the integration stage, the *fuser* can first measure the classifier's competence to provide an accurate decision, and later, as a result of weight modifications, the classifiers that are well adapted to the current concept are given the higher weight.

Overall, the *dynamic* weight modification (and hence, the classifier selection) provides many benefits in the data stream environment, especially when faced with *concept drift*. The classifier selection can be performed on different levels – for each incoming instance [209, 249], or the entire data batch depending on the identified concept [210]. The dynamic weight modification on an instance level becomes especially beneficial in the case of gradual drift when the instances in the time of a concept transition period belong to one of the two consecutive data distributions. The dynamic weight setup could offer the possibility of using old knowledge for a sample from a disappearing concept and new knowledge for a sample from an emerging one.

Pruning During the classifier selection, the outputs from some classifiers that were assessed as incompetent in a given region or for a given sample are not considered in the decision integration. While excluding a classifier during the selection phase is temporary and can be performed in each inference process, the *pruning* operation offers a possibility of a withdrawal of an incompetent classifier from the pool.

Pruning aims to reduce the number of classifiers, which can positively impact the *diversity*, and reduce the computational cost of using a hybrid system. For ensemble pruning, various strategies can be used:

- *Ranking-based methods* [161] – in this approach, classifiers are ranked based on their benefit to the overall ensemble. Often, measures of diversity or classification quality are used.
- *Clustering-based methods* [251] – first, classifiers are divided into groups, then pruning is performed in each created cluster. Keeping similar classifiers in a pool (those clustered into the same group) does not only increase the computational complexity of a method while not bringing benefits in terms of classification quality but can also decrease the method performance when the diversity of an ensemble is poor [38].
- *Optimization-based pruning* [177] – in this strategy, heuristic and evolutionary algorithms can be used to optimize measures of diversity and quality [21], computational complexity [186] or cost of feature acquisition [173].

In the case of data streams, pruning becomes highly beneficial when considering continuous adaptation to changing environments [128]. While dynamic selection could minimize the impact of incompetent classifiers, pruning can irrevocably remove them from the set. Pruning does not only improve the ensemble's diversity and accuracy but – what becomes essential in case of data streams – it allows for the maintenance of a low processing time.

A common mechanism used in hybrid methods for data stream classification is to limit the number of allowed ensemble members [128]. To allow for continuous training, the proposed solutions often incrementally update the pool of classifiers with the ones fitted on the new data [176]. Assuming an infinite volume of data stream, such solutions without a bounded ensemble size increase their computational complexity, compromising the first guideline of effective data stream processing, saying that the low and constant processing time is required from the method [55]. Pruning is the most straightforward method of maintaining a limited number of ensemble members.

2.4.3 Classifier ensembles for data stream processing

The use of *hybrid* classifiers for processing the data streams results directly from the possibilities of their incremental modifications [128]. From the point of ensemble construction – where the pool members can be iteratively trained with incoming samples or the new classifier instances can be added to the pool – to the possibilities of dynamic integration and adaptation of ensemble structure by classifier pruning, where hybrid methods offer many benefits when employed in data streams.

There are many criteria by which ensemble methods dedicated to data stream processing can be taxonomically categorized. The main criteria for taxonomically categorizing methods are:

- Methods dedicated to *chunk-based* or *online* processing [128].

  The scheme of sample arrival in the context of processing methods often affects the method architecture – especially in the context of single-time access to each input [55]. Therefore, the methods adapted to batch processing perform different actions than those suitable for online processing. However, it is worth keeping in mind that any solution dedicated to chunk-based processing can process an online stream when using a buffer, or, vice-versa, a data chunk can be divided into individual samples, later processed by a method for online data stream processing.

- Methods for processing *stationary* or *non-stationary* data streams [128].

Ensemble approaches suited for non-stationary data streams usually involve some dedicated solution for detecting concept drift and reacting to such change. Meanwhile, the solutions for stationary data streams only focus on incremental generalization of knowledge.

- Adaptation strategy – where methods can follow *active* or *passive* approach [4, 80, 128, 150].

  This is the most frequently mentioned division axes of currently used ensemble classification methods. According to work by Lindstrom et al. [150], methods can use *continuous rebuild* or *triggered rebuild* approaches. According to work by Krawczyk et al. [128] and by Agrahari and Singh [4], the methods are divided into *active* and *passive*. Gomes et al. [80] divided methods into *proactive* and *reactive* (or *blind* and *informed*, following the nomenclature from Gama et al. [71]).

- Strategies for modifying the *ensemble structure* and *integration rules*, described in more detail by Kuncheva [134].

  Within this criterion, methods were divided into five categories, which consider (*a*) knowledge updating methods (by adding members to the pool or by training individual experts), (*b*) structure modification methods (e.g., pruning), and (*c*) the methods of combining responses of individual team members. Kuncheva also considers the possibility of changing the significance of features during processing, which may result in a need to analyze new features without rebuilding the entire ensemble.

As mentioned, adaptation strategy is the most critical axis in terms of how methods adapt to concept drift. Passive methods update the classifiers incrementally, regardless of processing conditions, while active methods are those integrated with the mechanism of drift detection that update the ensemble in response to the recognized concept change or a drop in recognition quality [4]. The most important ensemble methods from these two categories are discussed below.

Passive adaptation Passive adaptation involves the continuous modification of ensemble members or the structure of an ensemble to passively adjust to changes visible in the data streams. The first ensemble classifiers for data streams followed the passive adaptation approach. Still, the majority of proposed methods belong to this category [128].

One of the earliest examples of ensemble methods adapted to data stream processing was the *Streaming Ensemble Algorithm* (SEA) [216], where successive classifiers were built on consecutive, disjoint data chunks, and later, incrementally joined the pool. In this approach, once the limit of classifiers in the pool was reached, the one with the lowest recognition quality measured on the current data was removed. Subsequent methods mainly focused on modifying the pruning criterion – removing a classifier from the ensemble – and proposed new integration mechanisms. *Accuracy Weighted Ensemble* (AWE) [229] additionally introduces the weighting of ensemble members' decisions based on their recognition quality. Therefore, the classifiers operating with poor quality have less influence on the final decision. The downside of AWE is the need for cross-validation to determine the weights of individual classifiers, which increases computational complexity. The only diversification approach in the above-mentioned methods is based on the assumption that subsequent data chunks in the stream are diverse. It is denoted as chunk-based approach [128].

The *Learn++.NSE* [58] algorithm extended the canonical online boosting [178] for non-stationary data streams. The ensemble method determined the weights of individual instances based on the errors made by the classifiers on the data input. The weights also determine the impact of individual classifiers. If a particular classifier performs poorly for the current data, its weight is reduced. In the case of recurring drifts, such a classifier can perform well again, and its weight can be restored as long as the classifier was not previously removed from the ensemble as a result of pruning.

In later works, the classifiers in the ensemble could be updated using new data without the need to build a new classifier from scratch. In *Dynamic Weighted Majority* (DWM) [116], classifiers are updated using newly incoming data, but adding new classifiers to the pool is possible. Ensemble pruning, adding new classifiers, and updating weights occur for every specific number of instances specified by the parameter. *Accuracy Updated Ensemble* (AUE) [35] used a similar scheme, but modified the method of determining weights for individual classifiers. In the *Weighted Aging Ensemble* (WAE) [239] method, the weight additionally depends on the estimated *age* of the classifier, calculated using the time spent in the ensemble and the performance of the particular model.

Since passive methods update the model regardless of the changes occurring in the stream, some operations may be redundant. On the other hand, relying on a specific module to detect concept changes may cause a lack of adaptation in case of non-recognized drift. The selection of a passive method may depend, therefore, on the ease of change detection (i.e., the frequency of system evaluation and the possibility of its manual monitoring), as well as on the consequences of the non-recognized change and possible implications of model adaptation at a slower-pace [176].

Active adaptation Active methods use an embedded drift detection mechanism [4]. In response to the detection, the method takes a series of actions to prevent the classification quality from dropping due to concept drift. These actions depend on the specific method but usually involve the initialization of a new classifier or a classifier pool [128]. A significant advantage of such methods is that they respond directly to changes and can maintain lower computational complexity. The potential weakness of this approach is the dependence on a drift detection quality of an external module. However, methods operating in this mode should ideally be resistant to incorrect, redundant signaling of the drift [128].

Diversity for Dealing with Drifts (DDD) [167] is an interesting approach using two sets of classifiers – the low-diversity and high-diversity ones. The low-diversity ensemble is used for recognition in a classification task. Meanwhile, the high-diversity one is only used during concept drift detection. Once the drift is detected, new sets of low-diversity and high-diversity classifiers are created to learn a new concept.

The *Aboost* [42] algorithm, inspired by the boosting approach, uses a statistical module responsible for detecting drifts. Identification of a drift leads to removing the classifiers included in the ensemble. Meanwhile, online bagging was extended to ADWIN *Bagging* and *Adaptive-Size Hoeffding Tree Bagging* [26] methods, using the ADWIN drift detector [24]. The same drift detector was also integrated with the *Leveraging Bagging Classifier* (LBC) [25]. This last approach modified the original online bagging by increasing resampling (sampling from $Poisson(\lambda)$ instead of $Poisson(1)$, where λ is a method hyperparameter) and using *output detection codes* to add randomization to the output of the ensemble.

The Poisson Distribution sampling is also used in modern methods, like *Kappa Updated Ensemble* (KUE) [38]. In KUE, classifiers are added to the pool only if they bring an overall benefit. Moreover, the decision of individual classifiers in the pool can be ignored in the integration procedure. Ensemble pruning is based on the *Kappa* measure, which provides an advantage in variable environments with drifting data streams and class imbalance.

As mentioned, active streaming ensemble algorithms employ some drift detection mechanisms. In this dissertation, the proposed methods for data stream classification belong to an active taxonomy branch and use metafeatures not only to detect the concept changes but also to extend the knowledge about the currently processed data.

2.4.4 Measuring classification quality

Almost every experiment requires an assessment of the quality of the proposed solution. In the case of classification, the quality of recognition can be assessed by various *simple* and *aggregated* measures, defined based on the confusion matrix for the binary classification case [240]. The choice of the quality assessment measure should take into account the imbalance ratio of the considered task.

Figure 2.17 presents the confusion matrix scheme. Specific cases are counted in the fields identified as: (a) True Positive (TP) – correctly recognized objects of the positive class, (b) False Negative (FN) – objects of the positive class incorrectly recognized as negative, (c) False Positive (FP) – objects of the negative class incorrectly recognized as positive, and finally (d) True Negative (TN) – correctly recognized objects of the negative class. Additionally, the sum of positive and negative objects is denoted as $P = TP + FN$ and $N = FP+TN$, and the sum of objects marked by the classification algorithm as positive and negative: $P' = TP + FP$ and $N' = FN + TN$, respectively.

Based on the counted cases, four simple metrics are calculated: (a) recall, (b) sensitivity, (c) precision, and (d) specificity, described by Equations (2.8 - 2.11).

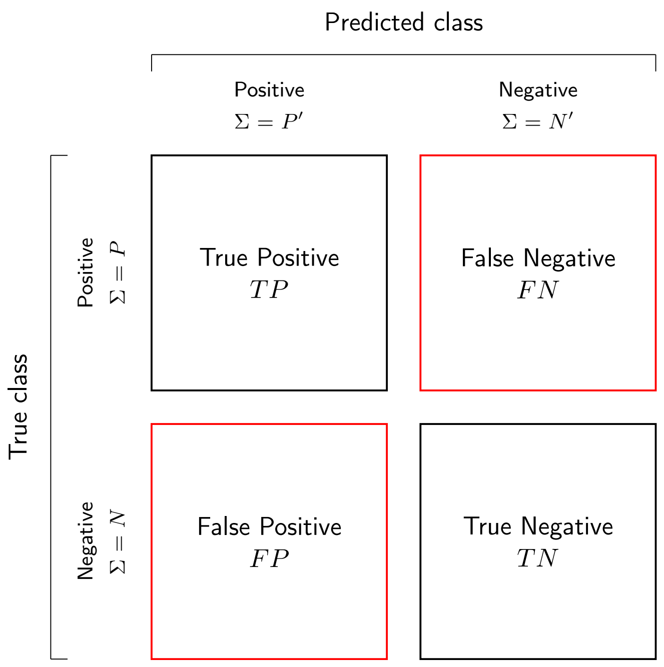

Figure 2.17: *The diagram presenting a confusion matrix for binary classification problem. Red fields indicate the sets of incorrect predictions – False Negatives and False Positives. The remaining two categories – True Positives and True Negatives – aggregate correct predictions.*

$$recall = \frac{TP}{P} \tag{2.8}$$

$$sensitivity = \frac{FP}{N} \tag{2.9}$$

$$precision = \frac{TP}{P'} \tag{2.10}$$

$$specificity = \frac{TN}{N} = 1 - sensitivity \tag{2.11}$$

Each of the presented simple metrics describes one specific criterion related to the recognition task, e.g., recall describes what share of positive objects was recognized correctly, while specificity – what share of negative objects was recognized correctly.

These basic metrics, in particular precision and recall, are essential for quality assessment in tasks with strong class imbalance, in which positive class objects constitute a minority class. Such a case is typical in medical diagnostics when there are usually fewer people with a positive diagnosis (sick) than with a negative diagnosis (healthy) [62].

For the balanced tasks, a frequently used metric is accuracy (ACC), described by Equation (2.12).

$$ACC = \frac{TP + TN}{P + N} \tag{2.12}$$

The accuracy measure describes the proportion of objects that were correctly recognized by the algorithm. However, this measure is not suitable for imbalanced problems due to the *accuracy paradox* [223]. As a result of evaluating a strongly imbalanced problem using the accuracy measure, in case of recall equal to zero (i.e., no correct recognition of any object of the minority class), the accuracy result may be equal to the percentage share of the majority class. Therefore, it is possible to achieve almost 100% accuracy, marking all objects as instances of the majority class.

In order to reliably evaluate imbalanced problems, among others, the F_β measure has been proposed, the most commonly used version of which is F_1 for the parameter $\beta = 1$. This measure is defined in Equation (2.13). Methods that achieve high results in precision and recall also achieve high results expressed by the F_β measure – hence, it is a measure seeking a compromise between these two simple metrics.

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{\beta^2 \cdot precision + recall} \tag{2.13}$$

In imbalanced data classification tasks, the Balanced Accuracy (BAC) metric is also often used, described by Equation (2.14).

$$BAC = \frac{recall + specificity}{2} \tag{2.14}$$

Balanced accuracy is defined as the average of recall and specificity for binary cases. In multi-class classification, it weights the accuracy according to the specific class share in the overall problem [88]. Unlike F_β, BAC is a symmetric measure [36] – that is, it is not directly dedicated to cases where the positive class is the minority. Hence, this metric is suitable for problems with time-varying levels of class imbalance, typical in *dynamically imbalanced streams* – when the initial minority class can become the majority class and vice versa.

2.4.5 Evaluation protocols for data stream classification

Data streams are characterized by potentially infinite volume, where samples inflow in the form of individual patterns – in online processing [106] – or in the form of data chunks – in offline or batch processing [128, 160]. The assumption that a complete set of processed data samples is available in the static form does not allow for a reliable assessment of the methods in the streaming environment [72]. Therefore, specific protocols are used to describe the real-world, incremental inflow of data and allow the assessment of classification tasks. Currently, there are four processing protocols for data streams [86].

Periodic holdout evaluation [72], presented in Figure 2.18, interleaves a portion of data and uses it to train the model. Then, the continuously incoming data samples from the stream are used for model evaluation at regular time intervals [73].

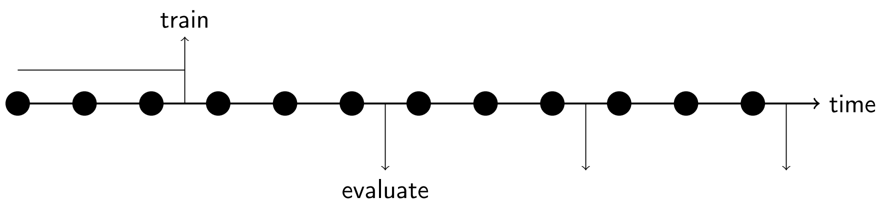

Figure 2.18: *The scheme showing the Periodic holdout evaluation protocol. The model is first trained on the interleaved set of samples and later evaluated at regular intervals.*

Prequential evaluation [73], presented in Figure 2.19, is an approach frequently used in online data stream processing, where the samples arrive individually, hence, there is no possibility to reliably evaluate the model based on a decision for a single sample. The *Prequential* approach uses a sliding window containing the most recent samples. In the case of the prequential error estimation, the window size or fading factor is of great importance in terms of model quality estimation.

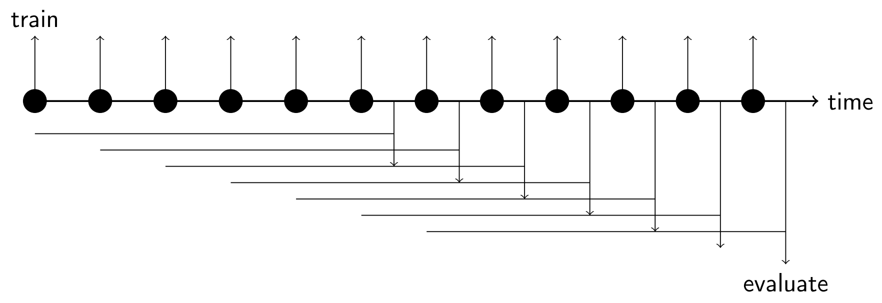

Figure 2.19: *The scheme showing the Prequential evaluation protocol. The model is evaluated with data contained in a sliding window of the most recent samples.*

Test-then-train evaluation [79], presented in Figure 2.20, is best suited for the data streams processed in the form of batches. Each incoming data chunk is first used to test the classification quality and later to train the model – except for the initial one, which is only used to fit the classifier. The reliable estimation of the method's quality can be possible due to the significant size of the data batch, playing a similar role to the window used in the prequential evaluation. An important difference between the prequential evaluation and *Test-then-train* is that the windows in the latter do not overlap.

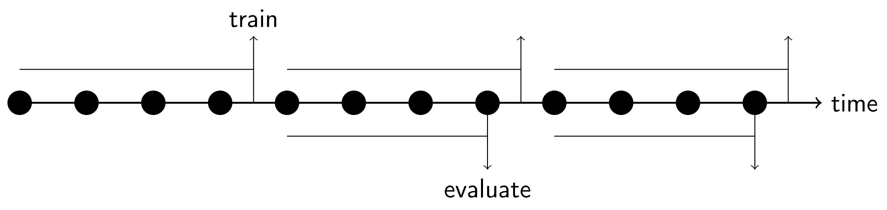

Figure 2.20: *The scheme showing the Test-then-train evaluation protocol. Each data chunk, apart from the first one, is used to evaluate the model and later to train it.*

Cross-validated Prequential evaluation [23], presented in Figure 2.21 for an illustrative scenario of three classifiers. This approach uses a set of k classifiers trained in parallel. For each incoming data sample, one classifier is randomly picked to test its performance, while all others are trained with the exact sample. To allow the evaluation of model quality, similarly to *Test-then-train* protocol, they need to first be partially fitted to the data. As typical in the online processing scenario, the prequential error is later computed for the sliding windows of recent samples or using a fading factor [49].

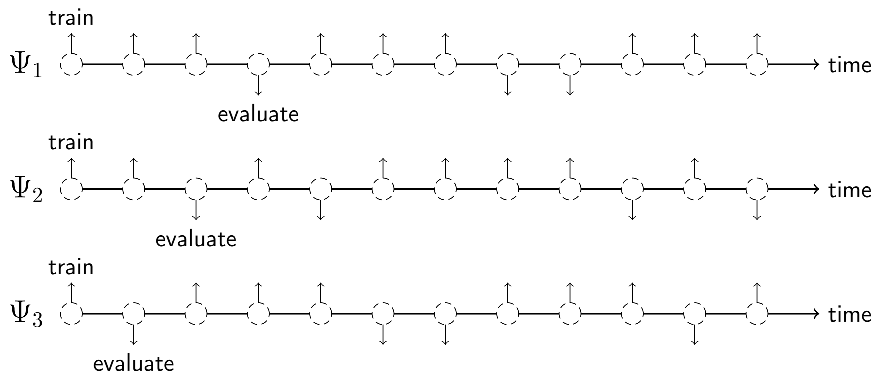

Figure 2.21: *The scheme showing the k-fold distributed Cross-validated Prequential evaluation protocol for an example of $k = 3$ classifiers. A set of models is trained and evaluated in parallel with available data samples. The errors are computed over a sliding window.*

There exist protocol modifications considering various strategies for selecting the trained and evaluated classifiers. In *split-validation*, only one classifier from the pool is trained using a newly appearing sample, contrary to *cross-validated* approach doing the opposite. In *bootstrap validation*, trained classifiers are selected based on the $Poisson(1)$ distribution. The presented *cross-validation* protocol makes maximum use of available samples.

In the presented research, all streams are processed as batches. Hence, the default experimental protocol used in most experiments is *Test-then-train*, where each batch is first used to test the model and estimate the recognition quality and later to train the solution.

Chapter 3

Concept drift detection

This chapter focuses on the *concept drift detection* task in the non-stationary data streams. The three included subsections introduce and describe the design of three novel methods for solving this task. The thorough description of each method is followed by the design of experiments and the critical analysis of their results.

The first method – *Complexity-based Drift Detector* [118] – uses the set of metafeatures describing the *complexity* of the classification problem [154] to study the changes visible in the data stream. While the original measures were proposed for the static datasets to assess the difficulty of the solved task, their usage in the data stream provides insight into the changes occurring in the processed concepts.

The following drift detector – *Statistical Drift Detection Ensemble* [126] – expands the *Statistical Drift Detection Method*, proposed by Micevska et al. [163], which analyzed the statistical metafeatures described by Webb et al. [230, 231]. The utilized measures, originally dedicated to the data streams, analyzed the probability distributions of the data, which, according to the definition of concept drift, should reveal those changes. The original method suffered from the *curse of dimensionality*, hence, the proposed approach used the ensemble learning paradigm to improve the quality of drift detection of the high-dimensional data.

The final method presented in this chapter – *Parallel Activations Drift Detector* [122] – addresses the problem of label availability in the data streams. The proposed approach synthesized the metafeatures of the data using the deterministic neural network, which allows for the unsupervised analysis of the data projections. The neural network outputs exhibit some dependence on the currently processed data distribution, allowing for the concept drift detection.

3.1 Complexity-based Drift Detector

This section describes the approach to concept drift detection using the classification *complexity measures* [154]. These measures, according to the current metafeature taxonomy [192], constitute a separate category intended to describe the difficulty of the classification task from various perspectives – from linear separability of classes and the sparsity of samples to the problem's imbalance ratio.

The *Complexity-based Drift Detector* (C2D) proposed in this section is a supervised ensemble method for concept drift detection. The operation of C2D is based on changes in the problem complexity measures over the stream processing flow. The proposed method belongs to the *implicit* drift detectors category – labels are required to calculate measures describing the problem complexity. Still, the detection itself is not based on the quality of recognition obtained by the underlying classification model, which, on the contrary, is typical for explicit methods. Figure 3.1 presents the motivation for using the problem complexity measures as a tool for concept drift detection.

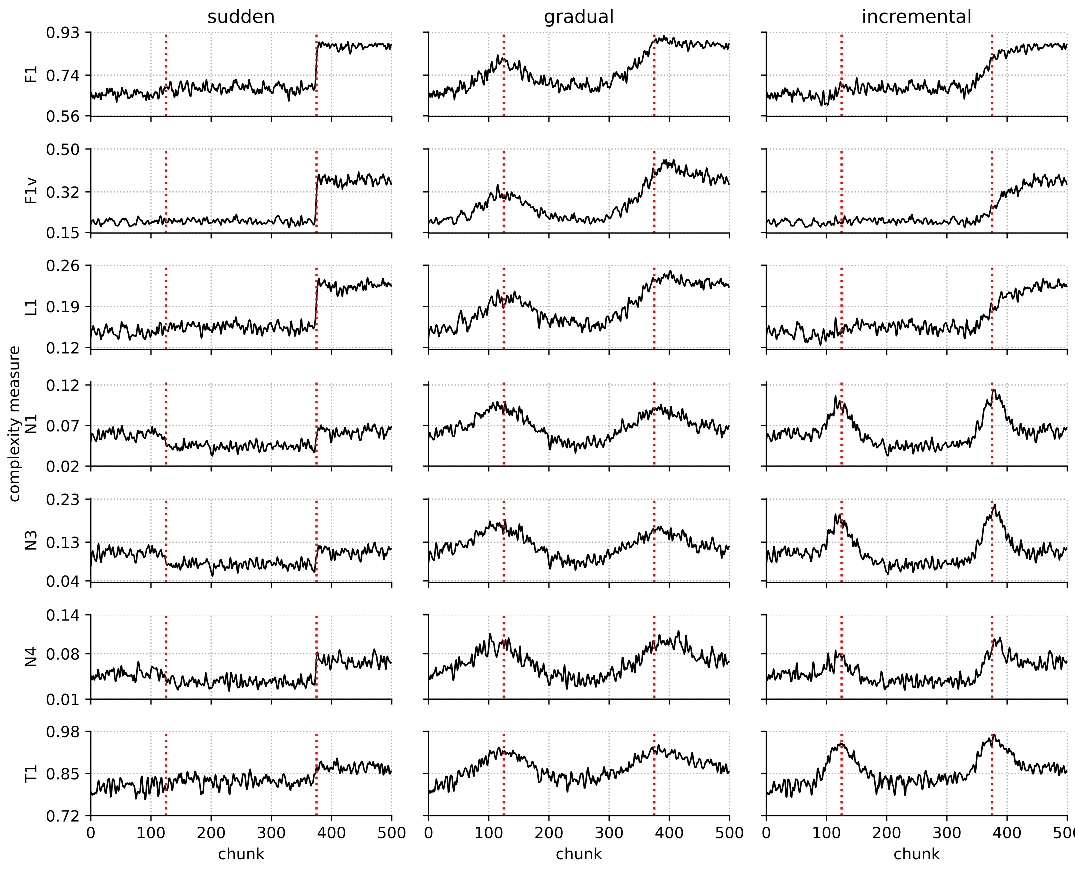

Figure 3.1: *Selected problem complexity measures calculated for the individual incoming chunks during the data stream processing. The black line shows the values of complexity measures smoothed with the Gaussian filter. The red vertical lines indicate the exact drift moments or the central points of its dynamics.*

The plots show the sampling of seven selected problem complexity measures in synthetic data streams consisting of 500 chunks, each containing 200 objects. The presented exemplary flows were described by sudden (left column), gradual (center column), and incremental drift (right column). The measures were determined individually for each data chunk.

Analysis of calculated measures shows that the measure values vary similarly to the concept changes' dynamic, depending on the drift type. The change occurs abruptly with sudden drift, while gradual and incremental drifts lead to smoother, slower changes. It is worth noting that not every concept drift is associated with a marked change in metric values. Depending on the nature of a given concept, drift may have a different impact on different measures. Therefore, the proposed approach uses a more sophisticated mechanism of distribution change detection – compared to the three-sigma rule in the baselines [24] – utilizing a pool of the one-class SVM classifiers. The final procedure of the proposed drift detection approach is presented in Algorithm 1.

Algorithm 1 Pseudocode of the *Complexity-based Drift Detector*

```
Input:
    $\mathcal{DS} = \{\mathcal{DS}_1, \mathcal{DS}_2, \dots, \mathcal{DS}_k\}$ – data stream,
                                                        ▷ Hyperparameters
    $measures$ – set of complexity measures
    $\theta$ – threshold
    $bf$ – bagging factor
    $e$ – limit of the classifiers in ensemble
    $\Psi$ – base one-class classifier
    $\Pi$ – ensemble of classifiers
                                                        ▷ Parameters
    $v$ – values of complexity metafeatures for current chunk
    $\mathcal{MF}$ – stored values of measures for past chunks

1: for all $\mathcal{DS}_k \in \mathcal{DS}$ do          ▷ Calculate complexity measures for current chunk
2:     for all $m_k \in measures$ do
3:         $v \leftarrow m_k$ for $\mathcal{DS}_k$
4:     end for                                        ▷ Select measure values from current concept
5:     $l \leftarrow$ number of chunks since last drift
6:     $X_M \leftarrow$ last $l$ values of $\mathcal{MF}$   ▷ Initialize the pool of classifiers with classifier fitted with all metasamples
7:     if $\Pi$ is empty then
8:         fit $\Psi$ with $X_M$
9:     end if                                         ▷ Randomly select metasamples according to bagging factor
10:    if $\Pi$ is not empty then
11:        $X'_M \leftarrow bf \cdot |X_M|$ random values from $X_M$
12:        fit $\Psi$ with $X'_M$
13:    end if                                         ▷ Extend the pool of classifiers
14:    add $\Psi$ to $\Pi$                             ▷ Calculate decision functions for detection
15:    $ms \leftarrow$ mean decision function of $\Pi$ for $v$   ▷ Drift detection
16:    if $ms < -\theta$ then
17:        signalize drift
18:        $ensemble \leftarrow \emptyset$
19:    end if                                         ▷ Ensemble prunning
20:    if $|\Pi| > e$ then
21:        remove oldest classifier from $\Pi$
22:    end if                                         ▷ Store metafeatures from current chunk
23:    store $v$ in $\mathcal{MF}$
24: end for
```

At the algorithm initialization, a set of problem complexity measures, described in Subsection 2.1.2, is specified as a hyperparameter of the method. Those measures are used for the purpose of the drift detection procedure. The measures taken into account during the processing may be adapted to the expected behavior – for example – to focus only on measures relating to the samples' neighborhood, sensitizing the method to the changes in the specific context.

For each data chunk, the values of the specified measures are calculated and saved to a vector v (line 3). Only the measurement values established since the last drift signaling are considered in the detection process (lines 5:6). Identification of atypical changes in the stream employs an auxiliary pool of one-class classifiers $\Pi = \{\Psi_1, \Psi_2, \dots, \Psi_e\}$. Those models are trained in a new *metaproblem space* defined by the considered *metafeatures* calculated for successive data stream batches.

Indication of *positive* class by a one-class classifier Ψ_k means that the chunk is recognized as dominated by objects from a concept significantly different from the one currently processed by the primary recognition model. The drift occurrence is signaled with a low decision function value, meaning a strong indication of an outlier batch. It should be emphasized that the one-class classifiers are trained with the values of problem complexity measures – i.e., the problem *metafeatures*. Their purpose is only to identify the concept unknown to the primary classification model – to solve a *metaproblem*, where the *metatarget* is the recognition of a current or a new concept.

The pool of one-class classifiers (Π) is an empty set during initialization and cleared after the drift signaling. The first classifier that enters the Π is trained using measures from all considered past chunks (lines 7:9). In subsequent iterations, the classifier to be included in the ensemble is trained using a subset of instances – randomly selected $bf \cdot |X_M|$ where $|X_M|$ indicates the number of available instances equal to the number of chunks since the last drift signaling (lines 10:13). The set X_M aggregates the metafeatures calculated for batches after the last concept drift.

The *bagging factor* (bf) is a hyperparameter determining the number of instances used to train one-class classifiers. Directly after detecting a drift, all classifiers from the ensemble are removed. For each chunk in the new concept, $100\% \cdot bf$ of all available instances are randomly selected and then used to train the classifier. Bagging is intended to increase the diversity of the classifiers in the pool. It can also reduce the training time due to the smaller number of samples used to train the classifiers in the pool. For illustration – for $bf = 1$, representations of all chunks from the current concept are used; for a hyperparameter of 0.5, representations of 50% of the chunks from the current concept are randomly selected to train the classifier added to the ensemble.

After extending the Π with an additional classifier, the method determines whether the current chunk belongs to a known concept. According to the measures calculated for the current chunk, decision function is collected from all ensemble members. A drift is signaled if their average decision function value falls below the negative θ hyperparameter (lines 15:19). The negative threshold is used due to the characteristic of the *one-class* SVM using the RBF kernel, which assigns negative decision function values to instances not belonging to the class under consideration. The drift signaling is associated with removing all classifiers from the pool (lines 17:18). If the maximum number of classifiers in the pool set by the input hyperparameter e has been exceeded, the oldest classifier is removed from Π (lines 20:21). After the detection procedure, the measure values for the current chunk v are saved to historical values $\mathcal{MF}$.

It should be noted that even if the limit of the classifier pool is 1, at least two one-class classifiers are in the pool – the model remaining from the previous batch and the new one – added for the considered chunk.

3.1.1 The design of experiments

This subsection describes the setup of experiments prepared to evaluate the C2D approach. The subsection describes the generated and selected data streams, the experimental environment, and the goals of specific experiments.

Data streams The two main experiments were conducted on synthetic data streams generated with a *stream-learn* library [133]. In order to stabilize the results, each selected stream configuration was replicated ten times with varying base concepts. Both *prior probability* and *conditional probability* concept drifts were considered with sudden dynamics of change or stretched over time (incremental and gradual drifts). The types of concept drifts have been divided into six different categories, presented in Table 3.1. The abbreviation of a stream visible in the first column describes them in the contexts

Table 3.1: *The configuration of concept drift types in evaluated streams. Six types of concept changes were defined, exhibiting changes in both prior probability (first aggregated column) and covariate conditional probability (second aggregated column).*

	PRIOR PROBABILITY			COVARIATE CONDITIONAL PROBABILITY		
ABBREVIATION	NONE	ABRUBT	SMOOTH	SUDDEN	INCREMENTAL	GRADUAL
BAL-SUDD	X			X		
BAL-INCR	X				X	
BAL-GRAD	X					X
IMB-SUDD		X		X		
IMB-INCR			X		X	
IMB-GRAD			X			X

of prior probability changes, with BAL indicating a balanced data stream without the drift affecting the prior probability distribution and IMB an imbalanced data stream with varying prior probability distribution. The second part of an abbreviation describes the drift dynamic.

In streams where both prior probability and conditional distribution drifts were present, the central points of these two drifts lie at the same time point to preserve their synchronous occurrence. Each generated stream comprised 2,000 chunks, each with 500 instances, leading to processing scenarios with one million examples. Each scenario contained the binary classification problem, described by 2 clusters per class and successively 8, 16, and 32 features. Over each stream, seven evenly spaced concept drifts were injected. The configuration of generated data was additionally presented in Table 3.2.

Table 3.2: *Generator configuration for synthetic data streams used for the experiments on* C2D *method. The first column describes the data characteristics or hyperparameters of the data stream generator, and the second column is their specified values.*

CHARACTERISTICS	CONFIGURATION
Number of chunks	2000
Chunk size	500
Number of features	8, 16, 32 (100% informative)
Drift frequency	7 drifts
Drift dynamics	*sudden*, *incremental*, *gradual*
Replications	10

After the preliminary studies, it was determined to take into account all of the classification complexity measures, except for the *Collective Feature Efficiency* measure ($F4$). The description of the complete set of those measures, along with the motivation for their usage, was presented in Section 2.1.2. The $F4$ measure excluded from the pool was characterized by large deviations regardless of the occurring drifts and caused redundant detection signaling. Ultimately, 21 complexity measures were used. The measures were calculated using the implementation from the *problexity* package, designed and developed for the purpose of the research presented in this dissertation [119].

For the final experiment, six real-world data streams were selected. Streams were processed in batch sizes of 250 instances. Some of the evaluated data streams were characterized by a significant class imbalance. In order to minimize the preprocessing of the data, the portions where only single-class instances were present were skipped during the processing. For those data batches,

Table 3.3: *The characteristics of selected real-world data streams. The first column presents the name of a dataset, and the following columns describe its characteristics.*

DATA STREAM	FEATURES	CHUNKS	PF$_{min}$	PF$_{max}$
`covtype-1-2vsAll`	54	453	0.004	0.976
`electricity`	8	182	0.132	0.708
`poker-lsn-1-2vsAll`	10	1217	0.004	0.940
`INSECTS-abrupt`	33	284	0.084	0.280
`INSECTS-gradual`	33	284	0.032	0.548
`INSECTS-incremental`	33	284	0.036	0.684

the utilized classification complexity
measures intended for drift detection were not possible to compute according to their definition [154].

The detailed description of selected real-world data streams is presented in Table 3.3. The columns describe the number of features, the number of chunks in the final data stream, and the range of problem imbalance ratio calculated within the chunks. The presented imbalance ratio is described by a fraction of positive samples (PF) in a given chunk, compared to the number of samples in a chunk, which as well can be interpreted as the *prior probability* of a positive class.

Hyperparameter optimization The first experiment aimed to select appropriate hyperparameters of the method. The evaluation searched for three values specifying the method operation:

- *limit of the classifiers* (e) – five examined values ranging from 1 to 20,
- *bagging factor* (bf) – values 0.25, 0.5, 0.75,
- *threshold* of detection signaling (θ) – 10 uniformly sampled values from 0.2 to 4.

Increasing the classifier limit (e) may become a factor in increasing the method's stability and making it resistant to noise in supporting the responses of individual classifiers, which, however, may also harm the time and memory complexity of the method. The *bagging factor* (bf) hyperparameter influences the ensemble diversity. At low values of bf, the classifiers have too little information about the current concept (they may be underfitted), which may as well harm the operation of the method. The drift signaling threshold (θ) should most significantly impact the method's performance. Setting the threshold too low can cause too frequent detections. On the other hand, setting it too high might lead to detections resulting only from accumulating patterns since decision function variance in *one-class* SVM often depends on the overall size of its training set, which was observed in the preliminary experiments.

Comparison with reference methods In the second experiment, after selecting the appropriate hyperparameters of the method for the tested processing scenarios, the operation of the proposed approach was evaluated in comparison with the *state-of-the-art* detectors. The C2D detector was compared with five reference methods, whose parametrization is described in detail in Table 3.4.

Table 3.4: *The configuration of reference drift detectors for comparison experiment with* C2D *approach. The first two columns specify the method acronym and its name, and the final column – its configuration.*

METHOD		CONFIGURATION
DDM	*Drift Detection Method* [70]	default detection *threshold* of 3, the base classifier used for error monitoring was Gaussian Naive Bayes; the *skip* hyperparameter was set to 30
EDDM	*Early Drift Detection Method* [12]	default *beta* of 0.9, the base classifier used for error monitoring was Gaussian Naive Bayes
ADWIN	*Adaptive Windowing* [24]	default *delta* of 0.002, the base classifier used for error monitoring was Gaussian Naive Bayes
HDDM$_A$	*Hoeffding Drift Detection Method* with *Bounding Moving Averages* [66]	the default *drift level* of 0.001, the base classifier used for error monitoring was Gaussian Naive Bayes
HDDM$_W$	*Hoeffding Drift Detection Method* with *Bounding Weighted Moving Averages* [66]	the default *drift level* of 0.001 and λ of 0.05, the base classifier used for error monitoring was Gaussian Naive Bayes

Two first reference approaches – DDM [70] and EDDM [12] signal detections based on the frequency of errors or the distance between errors made by the underlying classification method, respectively. The ADWIN [24] detector's operation is based on sliding windows of variable width. The final two reference methods, HDDM$_A$ and HDDM$_W$ [66], use Hoeffding inequality and moving averages or weighted moving averages. The hyperparameters of reference methods were set according to their default values in the *scikit-multiflow* library [170]. Even though those methods can signal a concept drift warning, only the actual detections were considered in this experiment. The use of Gaussian Naive Bayes (GNB) as the base classifier was motivated by its natural incremental learning ability while maintaining low processing time, as well as its sensitivity to concept changes – especially valuable for explicit drift detectors.

The three *drift detection error measures*, described in Section 2.3.3, were used to evaluate the detection quality in the first and second experiments.

Real-world data stream processing The final experiment was intended to present the operation of the method on real-world data streams. Since the exact timing of the concept change in the stream remains unknown, it is not possible to evaluate the method using the measures of drift detection errors. Therefore, only the moments of detection and the classification complexity measures variability are presented in this experiment. Such an analysis does not allow the complete evaluation of method performance in terms of specific measures. However, it can be used for an *interpretation* of a detection – showing the impact of changes occurring in the data on the selected classification complexity measures.

3.1.2 Experimental evaluation

This subsection presents and discusses the results of the experiments performed on the proposed C2D method. The analysis highlights the dependencies of the method's hyperparameters on the detection quality, offering the proper configuration of the method. After the hyperparameter selection, the method was compared with the reference drift detectors, presenting the visual analysis of the detection moments, the drift detection error measures, and statistical analysis of the results. Finally, C2D was used to analyze the real-world data streams, allowing to *interpret* the detection moments.

Hyperparameter selection The first experiment aimed to select appropriate hyperparameters of the method. The detection quality was examined in terms of three measures of drift detection error. The average results for these three criteria and a single stream are shown in Figure 3.2. It presents results for an 8-dimensional balanced stream with sudden concept drifts.

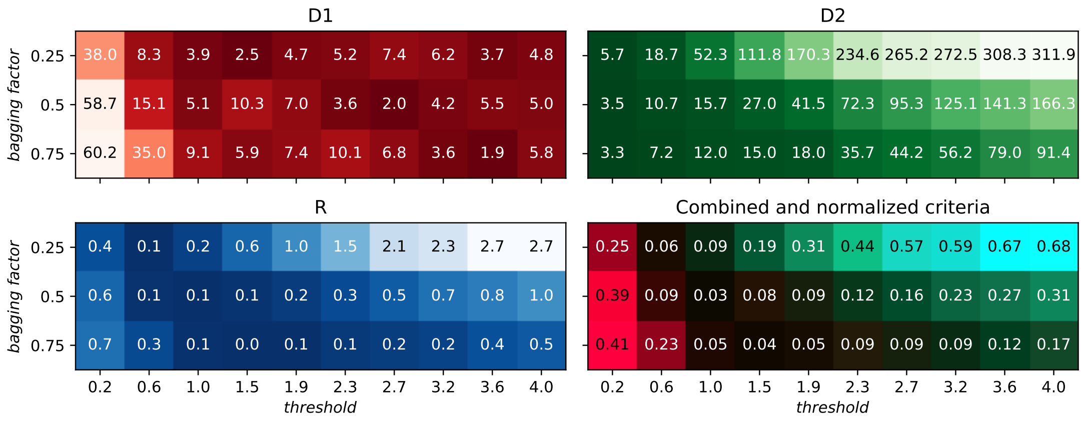

Figure 3.2: *Hyperparameter impact on specific drift detection error measures and combined criteria. The presented figures show average results for 8-dimensional balanced data streams with sudden concept drifts.*

The first three heatmaps present individual criteria in various color-maps, where the intensity depends on the average results – a light color indicates a high value of a given error, and the dark, saturated color – a low error. The figure presents, respectively, $D1$ – the average distance from each detection to the nearest drift in reds, $D2$ – the average distance of each drift to the nearest detection in greens, and R – the adjusted ratio of the number of drifts to the number of detections in blues. Since the measures describe the distance between drifts and detections and the ratio of their cardinality, the raw values of measures can extend to infinity and are bounded to zero in case of an ideal detection. The axes of the heatmaps present two out of the three tested hyperparameters. Only the analysis for the *bagging factor* (bf) and *threshold* (θ) hyperparameters, since for the last one (limit of the classifiers in an ensemble e), no statistically significant differences in the method's effectiveness were observed. The last cell shows the combination of the three tested criteria as an RGB image, where the subsequent color channels correspond to the specific errors: $D1$ as a red color channel, $D2$ as the green color channel, and R as the blue. The specific cells express the mean value of normalized error measures. The value of 0.0 could be interpreted as the lowest score across all error measures for the specified hyperparameters. The darkest parts of the heatmap mean the lowest error values (across all three measures), while saturated colors describe a high deviation between errors. The analysis of individual measures shows that individual criteria complement each other – minimizing a single error measure is not enough to indicate the correct operation of the method.

The far left side of each heatmap is associated with a low θ value and frequent redundant detections – small changes in metafeature values are enough for the method to signal a change of concept. This behavior is associated with high values of the $D1$ error measure, in which the distance to the nearest drift is calculated for each signaled detection. The error value is significant after averaging the values for all (including redundant) detections. Redundant detections do not have such a negative impact on the values of the $D2$ measure, in which only one – the closest detection – is taken into account for each drift that truly occurs. In the context of the R measure, which looks at the number of drifts in relation to the number of detections, redundant detections have some impact on the error values – hence the slight increase seen at a θ of 0.2.

On the contrary, the far right side of the graph, associated with a high θ, indicates few or no detections. The case of no detection makes it impossible to calculate the $D1$ and $D2$ measures, as the detection distances are undefined. The R measure cannot be calculated due to the presence of the number of detections in the denominator. In this extreme case, the drift detection error measures are calculated analogously for the detection signals in each data chunk. The case of few detections – e.g., the method signaling only one drift in the presence of seven, as in the streams analyzed in the experiment – results in a high $D2$ value, in which the distance to the nearest detection is calculated for each drift. Too few detections do not affect $D1$, as only the distance to the nearest drift is calculated for this exemplary single detection. In the case of the R measure, a significant increase in the metric value is observed, resulting from a larger value in the fraction numerator (the number of drifts is greater than the number of detections) when calculating the metric value.

Once these three measures are combined, all criteria can be considered when selecting the best hyperparameters. In the case presented in the figure, high values of the $D2$ and R measures translate into a cyan shade in the upper right corner of the heatmap –

for high values of the θ and low bf. The red area in the lower left corner is associated with a high $D1$ error for low values of the θ and high bf.

The results for all studied streams – for various dimensionality and drift types – are presented in Figure 3.3. Similarly, successive values of the θ are presented on the horizontal axis of heatmaps, while the bf values are represented on the vertical axis. The respective columns show the stream dimensionality, and the respective rows show various types of drift. The first three rows show the analysis for streams with a constant prior probability, while the last three – results in streams with additionally injected prior probability drift.

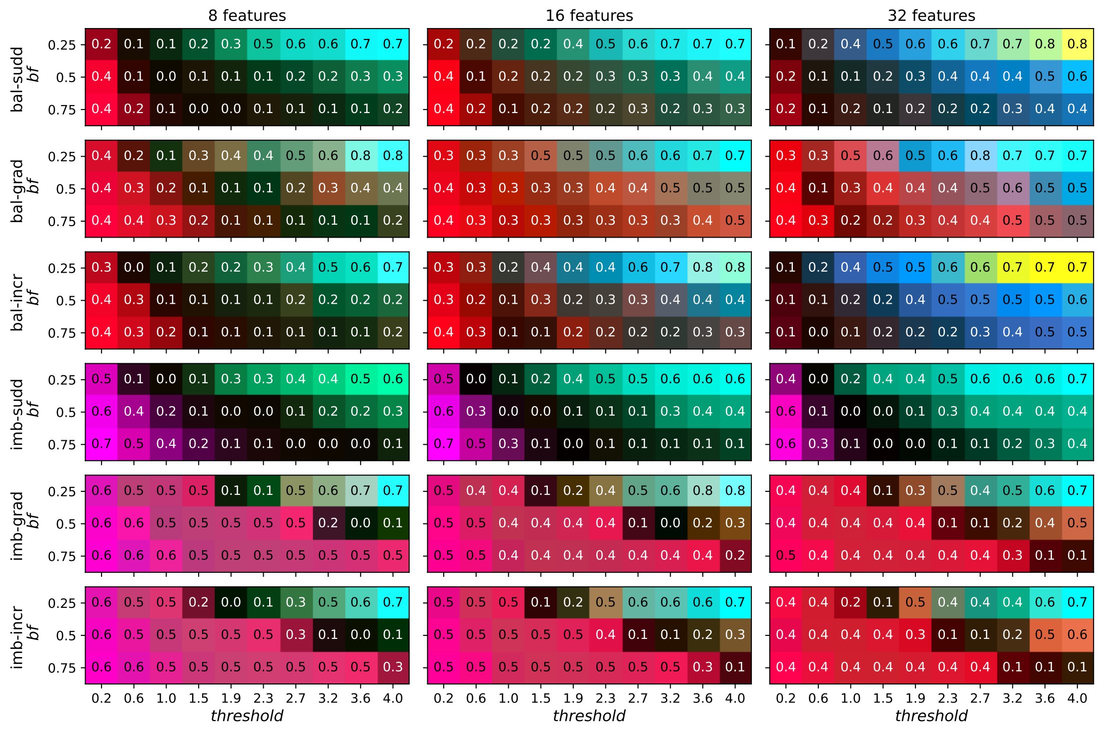

Figure 3.3: *The complete results of a first experiment of* C2D *method, showing the normalized mean drift detection error measures for tested hyperparameters. The darkest areas of heatmaps describe the lowest errors across all three measures.*

The hyperparameter's influence on the drift detection quality depends highly on the injected drift types. For each tested bf, it can be observed that the suboptimal value of the θ is different. Figure 3.3 shows that the broadest range of the threshold hyperparameter, which brings satisfactory results, usually occurs for the highest tested $bf = 0.75$. Therefore, this value has been selected as a default hyperparametrization of C2D. For the *threshold* hyperparameter, the default value of $\theta = 1.5$ has been selected.

Figure 3.4 shows precise stream flows for selected hyperparameters. Again, the columns represent the streams with various numbers of features, while the rows are associated

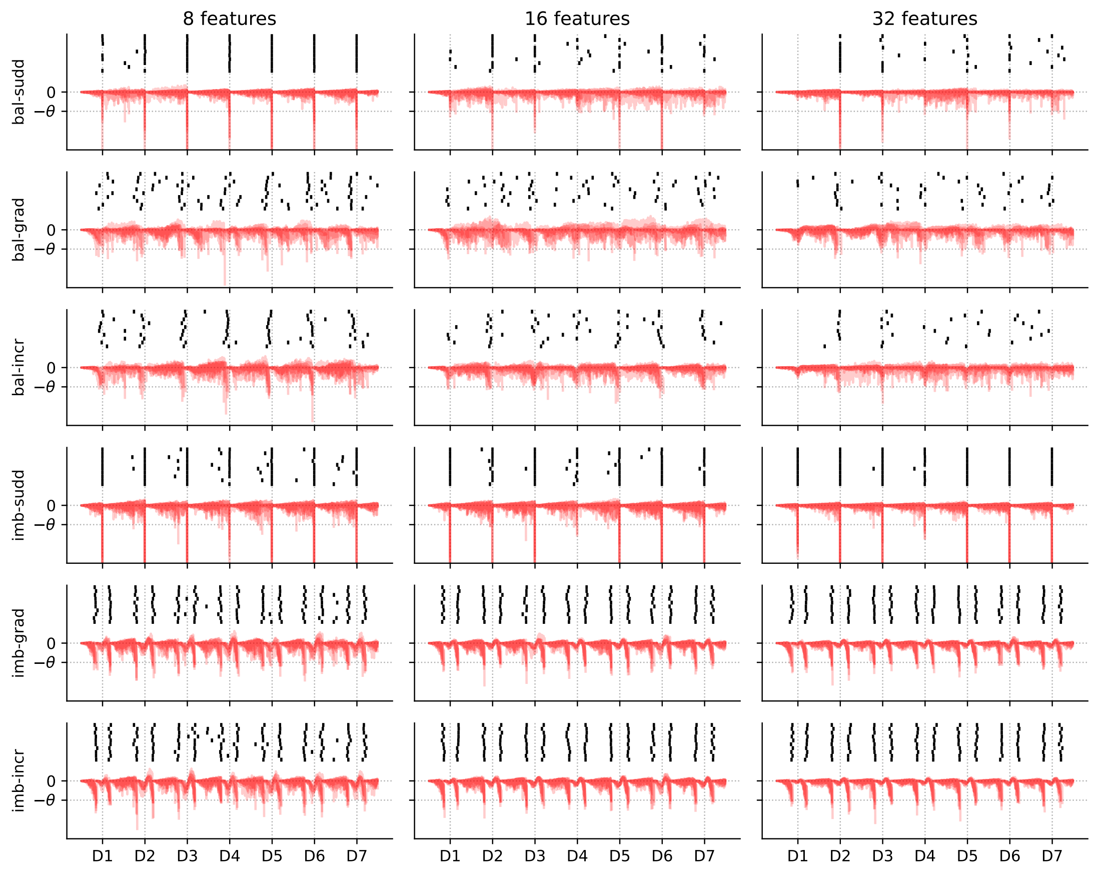

Figure 3.4: *Visualization of detection moments and decision function values for selected hyperparameters. The red plot indicates the decision function across replications, and the black points – the detection moments, signaled when the function falls below $-\theta$.*

with the drift types. Each drift detection in each of the ten replications is represented as a black point. The average values of the decision function of the one-class classifiers during processing are shown in red. Observation of the figure shows an unequivocal negative decision function spike during each drift detection, especially noticeable for sudden drifts.

It is worth noticing the dual signaling of each drift in the case of the last two types of streams with a smooth prior probability change. In these scenarios, the stream was balanced at the central drift points, as there was a transition from the previously minority class to the majority class and vice versa. For binary streams, this meant a gradual increase in the number of minority class samples up to the point of balance in the central moment of the drift. Then, the instances of the class that constitute the majority occurred less frequently. This behavior had a dynamic impact on the difficulty measures correlated with the imbalance ratio, particularly the measures $C1$ and $C2$. The first drift was signaled by the method when the number of samples of the minority class began to increase. The algorithm was so sensitive to these changes that it reacted with detection before the problem became balanced. The second detection appeared when, from chunks with a slight disproportion between classes, the problem became gradually more imbalanced. This behavior is not observable when the prior probability drift is sudden.

In processing dynamically imbalanced data streams, signaling each concept drift twice negatively impacts all the detection error measures, particularly the R measure, which is the adjusted drifts-to-detections ratio. However, in the case of a classification task, this method's behavior should not be considered unfavorable, as the algorithm sees a short-lived transitional concept that occurs at the moment of transition from one concept to another. In real applications, the method would detect more subtle changes in the concept. It is worth mentioning again the work by Bifet [22], showing that frequent, naive rebuilding of the model as a consequence of simulated detection can bring the highest classification quality. Therefore, resorting to frequent detections increases the benefits in terms of recognition quality but significantly increases the processing time, taking into account retraining the classification model. Thus, dual signaling of incremental and gradual drift by the C2D method is a mistake in the context of precise drift placement, which, however, may bring the desired effect of recognizing the transient concept and specializing the classifier for processing in the transition from one concept to another.

For streams described by a higher number of features, drift detection becomes more challenging, especially for balanced problems. Therefore, as typical for high-dimensional data [19], processing streams with more features become more complex and produce more significant errors.

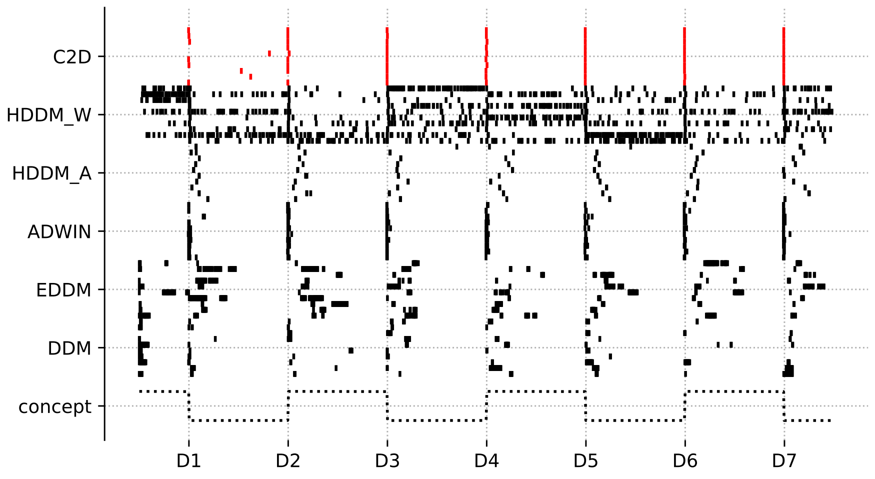

Figure 3.5: *Sample plot presenting the detection moments for a balanced stream with eight features and sudden concept drifts indicated on x-axis. The proposed approach is shown in red. Each point indicates a single concept change detection across ten experiment replications. The concept change dynamics is illustrated in the bottom with a dotted line.*

Comparison with reference methods The second experiment aimed to compare the method with *state-of-the-art* concept drift detectors. Figure 3.5 shows the experiment results for a single balanced stream with eight features with sudden concept drifts, following the visual assessment strategy presented in Subsection 2.3.3.

Individual methods are presented in rows, sequentially presenting detections performed by the C2D method, followed by the HDDM, ADWIN, EDDM, and DDM methods. In the last row, the dynamics of concept changes are marked with a dotted line, and the moments of drift on the horizontal axis are additionally marked with ticks. To emphasize the presented approach, the detections of C2D are shown in red. Each point in the figure represents a single detection made by the method in a single stream replication. Detections within the replication were placed one below the other so that the ideal operation of a given method would be equivalent to the presentation of single vertical lines coinciding with the moments of drift occurrence. Any individual points indicate redundant or delayed detections.

Figure 3.6 shows the results for the entire experiment configuration similarly. As in the first experiment, the columns show stream dimensionality, while rows indicate different concept shift scenarios. Compared to the reference methods, the C2D approach reports drift less frequently. The HDDM_W and ADWIN methods are characterized by a large number of redundant detections, especially in streams with incremental and gradual drift, both in the case of balanced and imbalanced drift streams. In most *hybrid* approaches, drift detection is associated with rebuilding the classifier using samples from the new concept. Thus, redundant signaling may cause unnecessary retraining of the classifier and negatively impact the time efficiency, which is an important factor when evaluating data stream processing. The presented evaluation strategy allows for an indirect estimation of time complexity of a recognition system by evaluating the R quality measure – with many redundant detections impacting both the processing time and the drift detection error measure.

The most outright detections appeared for streams with sudden drift, mainly when the *covariate conditional distribution* drift was connected with the same changes in the *prior distribution.* In the case of the last two types of drift (imbalanced gradual and imbalanced incremental), it can be seen that the detectors ADWIN and HDDM_W signal a change throughout the drift. In contrast, HDDM_A usually signals the final drift phase, while DDM and EDDM – the beginning of concept change. All reference methods repeatedly signal a shift during the transition. The C2D method behaves differently, signaling the beginning and end of the concept change for almost every transition. This behavior can be understood as recognizing a transient, temporary concept at the time of drift. Such

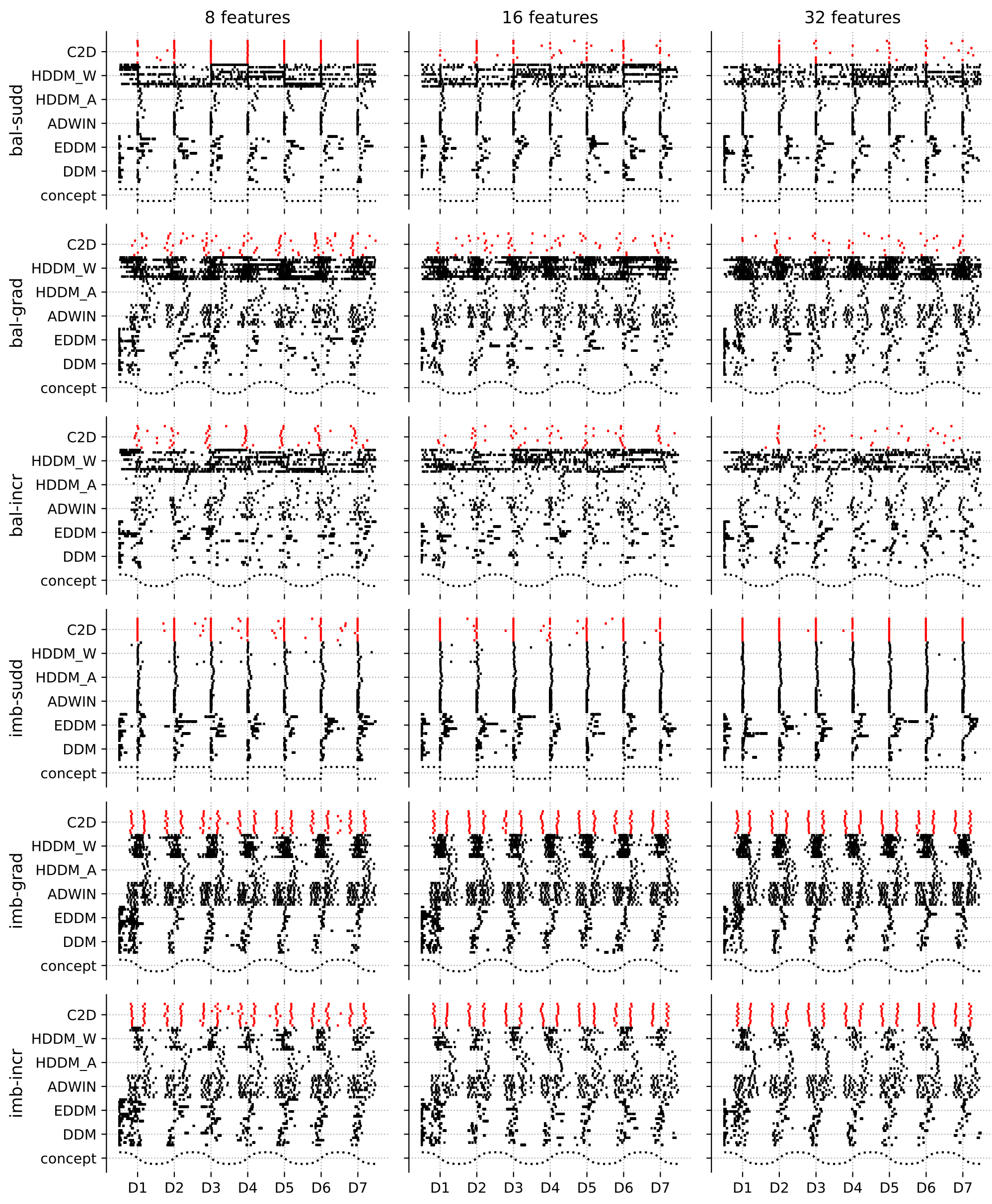

Figure 3.6: *Comparison with reference drift detectors. The columns present the results for various dimensionalities of the stream and the rows for various drift dynamics. Each detection is indicated with a single point, with the proposed* C2D *shown in red.*

desired behavior allows the classifier's adaptation to the processing of the transitional concept, reducing recognition errors in the transition period between concepts.

The drifts combining both prior and conditional changes are easier to recognize. In implicit methods, like the proposed C2D, this comes from the correlation between the problem complexity measures and the imbalance ratio. For the explicit methods, the change in the imbalance ratio only indirectly affects the frequency of errors made by the classifier. It may extend the timespan to propagate the information about changes in prior probability distribution.

Table 3.5: *Results of comparison experiment showing the mean D1 drift detection error measure. The columns present the results obtained by a specific drift detection method, while the rows represent the specific configuration of the generated stream.*

		DETECTION FROM NEAREST DRIFT (D1)					
		DDM (1)	*EDDM* (2)	*ADWIN* (3)	$HDDM_A$ (4)	$HDDM_W$ (5)	*C2D* (6)
8 FEATURES	BAL-SUDD	48.246	70.720	**2.578**	35.615	62.205	**5.140**
		2	—	**1 2 4 5**	2 5	—	**1 2 4 5**
	BAL-GRAD	61.986	53.575	51.808	88.074	51.361	**50.429**
		4	4	4	—	4	**1 4**
	BAL-INC	59.010	55.509	38.217	86.836	64.776	**30.727**
		4	4	1 2 4 5	—	4	**all**
	IMB-SUDD	29.198	64.990	**2.826**	14.446	11.063	21.094
		2	—	**all**	1 2 6	1 2 6	2
	IMB-GRAD	50.882	45.624	54.779	76.420	**25.696**	57.880
		4	4 6	4 6	—	**all**	4
	IMB-INC	45.725	43.102	50.921	78.614	**26.859**	56.294
		4 6	4 6	4 6	—	**all**	4
16 FEATURES	BAL-SUDD	32.321	64.173	**2.012**	29.010	61.095	17.332
		2 5	—	**all**	2 5	—	2 4 5
	BAL-GRAD	44.218	59.476	55.689	84.155	**50.321**	55.848
		4	4	4	—	**3 4**	4
	BAL-INC	**45.722**	70.092	**37.986**	94.507	63.937	**31.139**
		2 4 5	4	**2 4 5**	—	4	**2 4 5**
	IMB-SUDD	32.359	65.821	**2.815**	12.033	10.373	9.986
		2	—	**all**	1 2	1 2	1 2
	IMB-GRAD	53.477	38.508	55.668	72.185	**25.656**	57.650
		4	3 4 6	4	—	**all**	4
	IMB-INC	46.149	39.380	54.555	81.313	**26.664**	56.300
		4	3 4 6	4 6	—	**all**	4
32 FEATURES	BAL-SUDD	31.934	72.097	**2.042**	31.118	58.303	19.990
		2 5	—	**all**	2 5	2	2 4 5
	BAL-GRAD	**46.754**	54.382	56.862	85.123	**46.822**	**43.998**
		3 4	4	4	—	**3 4**	**3 4**
	BAL-INC	34.288	66.909	37.997	105.545	65.033	55.550
		2 4 5 6	4	2 4 5 6	—	4	2 4
	IMB-SUDD	27.338	67.478	**2.889**	9.936	5.623	**2.146**
		2	—	**1 2 4 5**	1 2	1 2 4	**1 2 4 5**
	IMB-GRAD	47.783	35.292	54.989	74.890	**24.524**	56.000
		4	3 4 6	4	—	**all**	4
	IMB-INC	45.843	38.632	52.612	87.449	**27.172**	56.021
		4	3 4 6	4 6	—	**all**	4

The proposed method for the selected threshold does not present ideal results for streams of large dimensionality. Reducing its value could improve the detection quality for balanced streams with incremental and gradual drift but would result in redundant drift signaling for streams with lower dimensions.

Tables 3.5–3.7 show the results measured with *drift detection errors* for the tested methods and streams. The table includes the paired Student's T-test results with the significance level $\alpha = 5\%$. The table cells present the mean result across experiment replications, and below the mean metric values, the indexes of the methods that the given one is statistically significantly outperforming. The results with a statistically significant advantage over most methods are emphasized in bold. Additionally, the column presenting the results for C2D is highlighted in red.

In the $D1$ criterion, describing the average distance from each detection to the nearest

Table 3.6: *Results of comparison experiment showing the mean D2 drift detection error measure. The columns present the results obtained by a specific drift detection method, while the rows represent the specific configuration of the generated stream.*

		DRIFT FROM NEAREST DETECTION (D2)					
		DDM (1)	$EDDM$ (2)	$ADWIN$ (3)	$HDDM_A$ (4)	$HDDM_W$ (5)	$C2D$ (6)
8 FEATURES	BAL-SUDD	94.829	68.686	**1.000**	37.957	**1.557**	**14.343**
		—	—	**1 2 4**	1 2	**1 2 4**	**1 2 4**
	BAL-GRAD	132.614	62.529	15.743	103.129	**3.057**	41.900
		—	4	1 2 4 6	—	**all**	4
	BAL-INC	163.957	72.714	**10.771**	110.157	**9.700**	27.200
		—	4	**1 2 4 6**	—	**1 2 4 6**	1 2 4
	IMB-SUDD	10.857	35.171	**0.129**	13.700	4.786	**1.686**
		2	—	**2 4 5**	2	2 4	**2 4 5**
	IMB-GRAD	87.357	10.414	11.714	67.257	**3.171**	49.343
		—	1 4 6	1 4 6	—	**all**	4
	IMB-INC	34.200	20.029	11.471	74.743	**6.786**	47.629
		4	4 6	4 6	—	**all**	4
16 FEATURES	BAL-SUDD	56.971	40.871	**0.271**	29.357	1.357	77.857
		—	6	**all**	2 6	1 2 4 6	—
	BAL-GRAD	213.929	34.457	17.557	89.900	**2.743**	77.000
		—	1 4 6	1 2 4 6	—	**all**	—
	BAL-INC	221.657	53.343	**12.500**	103.386	**9.671**	67.657
		—	4	**1 2 4 6**	—	**1 2 4 6**	4
	IMB-SUDD	13.171	39.043	**0.071**	11.471	5.157	3.629
		2	—	**1 2 4 5**	2	2 4	2 4
	IMB-GRAD	60.214	13.000	12.100	61.257	**2.329**	54.714
		—	4 6	4 6	—	**2 3 4 6**	4
	IMB-INC	37.486	20.743	13.914	79.729	**7.257**	53.543
		4	4 6	1 2 4 6	—	**all**	4
32 FEATURES	BAL-SUDD	26.729	45.343	**0.429**	30.143	0.957	129.571
		6	6	**all**	2 6	1 2 4 6	—
	BAL-GRAD	84.386	34.586	16.471	86.357	**2.486**	98.914
		—	4 6	1 2 4 6	—	**all**	—
	BAL-INC	45.357	49.257	**11.214**	109.629	**8.671**	164.814
		4 6	4 6	**1 2 4 6**	6	**1 2 4 6**	—
	IMB-SUDD	8.214	41.086	**0.029**	9.343	3.757	1.529
		2	—	**1 2 4 5**	2	2 4	1 2 4
	IMB-GRAD	83.029	12.271	11.900	60.471	**2.886**	52.714
		—	4 6	4 6	—	**2 3 4 6**	4
	IMB-INC	56.286	20.129	12.157	86.914	**7.886**	51.371
		—	4 6	1 2 4 6	—	**all**	4

drift (Table 3.5), the best results are achieved by the ADWIN method for sudden drifts, HDDM$_W$ for incremental and gradual drifts, and C2D for low-dimensional and balanced streams. Since C2D signals drifts in streams with varying imbalance ratios only at the beginning and end of the concept change period, as can be seen in Figure 3.6, the detections may be distant from the drift center point. HDDM$_W$ and ADWIN achieve better values for this error measure, as they remain active for almost the entire duration of the concept change with incremental and gradual drift.

In the $D2$ criterion, describing the average distance of each drift to the nearest detection (Table 3.6), similarly, the C2D and HDDM$_W$ methods achieve the best results. With the very high activity of these methods during the concept transition, the average distances from each detection to the nearest drift (analyzed in $D1$) and from each drift to the nearest detection (analyzed in $D2$) are small, translating into low error values.

Table 3.7: *Results of comparison experiment showing the mean R drift detection error measure. The columns present the results obtained by a specific drift detection method, while the rows represent the specific configuration of the generated stream.*

		RATIO OF DRIFTS TO DETECTIONS (R)					
		DDM (1)	*EDDM* (2)	*ADWIN* (3)	*HDDM$_A$* (4)	*HDDM$_W$* (5)	*C2D* (6)
8 FEATURES	BAL-SUDD	0.786	0.914	0.696	0.458	0.912	**0.033**
		2 5	—	1 2 5	1 2 3 5	—	**all**
	BAL-GRAD	0.760	0.900	0.802	0.392	0.955	**0.264**
		2 5	5	2 5	1 2 3 5	—	**all**
	BAL-INC	0.776	0.903	0.692	0.308	0.913	**0.072**
		—	—	2 5	1 2 3 5	—	**all**
	IMB-SUDD	0.889	0.936	0.853	0.488	0.543	**0.236**
		2	—	1 2	1 2 3 5	1 2 3	**all**
	IMB-GRAD	0.895	0.899	0.884	0.600	0.909	**0.529**
		—	—	5	1 2 3 5	—	**all**
	IMB-INC	0.912	0.930	0.834	**0.527**	0.794	**0.530**
		—	—	1 2	**1 2 3 5**	1 2	**1 2 3 5**
16 FEATURES	BAL-SUDD	0.825	0.899	0.725	0.476	0.905	**0.337**
		—	—	1 2 5	1 2 3 5	—	**all**
	BAL-GRAD	0.674	0.895	0.797	0.437	0.952	**0.160**
		2 5	5	2 5	1 2 3 5	—	**all**
	BAL-INC	0.699	0.883	0.657	**0.361**	0.906	**0.238**
		5	—	2 5	1 2 3 5	—	**1 2 3 5**
	IMB-SUDD	0.888	0.922	0.862	0.476	0.511	**0.097**
		2	—	2	1 2 3	1 2 3	**all**
	IMB-GRAD	0.895	0.890	0.881	0.613	0.916	**0.500**
		—	5	5	1 2 3 5	—	**all**
	IMB-INC	0.906	0.903	0.823	**0.505**	0.792	**0.500**
		—	—	1 2	**1 2 3 5**	1 2 3	**1 2 3 5**
32 FEATURES	BAL-SUDD	0.841	0.899	0.693	**0.473**	0.896	0.820
		5	—	1 2 5	**all**	—	—
	BAL-GRAD	0.705	0.879	0.793	**0.444**	0.953	**0.442**
		2 5	5	2 5	**1 2 3 5**	—	**1 2 3 5**
	BAL-INC	0.837	0.909	0.560	**0.345**	0.888	1.042
		2	—	1 2 5 6	**all**	—	—
	IMB-SUDD	0.887	0.924	0.862	0.500	0.515	**0.013**
		2	—	1 2	1 2 3	1 2 3	**all**
	IMB-GRAD	0.800	0.894	0.879	0.640	0.918	**0.500**
		5	—	5	1 2 3 5	—	**all**
	IMB-INC	0.867	0.893	0.822	**0.502**	0.744	**0.500**
		—	—	1 2	**1 2 3 5**	1 2 3	**1 2 3 5**

The last criterion R describing the adjusted ratio of the number of drifts to the number of detections is shown in Table 3.7. Too many detections result in error values converging to 1, and too few – in a measure exceeding the value of 1 and extending to infinity in case of very few detections compared to the number of actual drifts. In the case of the comparative experiment, the R error values exceeding 1 appeared only for the C2D method and incremental drift in the balanced stream described by 32 features. As noted earlier, the C2D method for this type of stream signaled less drift than actually occurred in the stream. For the ADWIN and HDDM methods, which detected significantly more drifts than were present in the stream, the error values are significant but not exceeding 1. Except for the two cases, the lowest error values are achieved by the C2D method.

The presented measures of detection errors should never be considered individually, as they constitute a complementary whole that describes the quality of the detector's operation. In measure $D2$, the method reporting drift for each data chunk would achieve zero error values. Respectively, redundant detections near the drift do not harm the $D1$ measure. The detection density near the drift reduces the impact of outlier detections on the error's value. The optimal values in the R criterion are achieved by a method that signals exactly as many detections as there are drifts, regardless of the moment of their signaling. The evaluation of drift detection methods should consider the bias of all detection error measures and as well use visual analysis.

The visual presentation of detection moments in Figure 3.6 is particularly beneficial when trying to understand the operation of the detection mechanism and its impact on the *drift detection error* measures. The figure reveals the indication of the temporary concept by C2D in the case of drifts including prior probability change. This particular behavior of the method harms the analyzed *drift detection errors* measures. However, it may become useful in real-world applications, when reacting to any changes in the stream may be of great importance for the data recognition quality.

Real-world data stream processing Figure 3.7 shows the result of processing real-world data streams. The chunk numbers are shown on the horizontal axis of each subplot. The red plot represents the mean decision function values of the one-class classifiers responsible for the detection process. Black vertical lines mark the moments of drift signalization – when the value of the decision function exceeds the negative value of the θ. For the *poker-lsn* stream, it was set to 1.5, while for the other streams – to 0.75.

In this experiment, the operation of the drift detector cannot be unequivocally assessed because of the lack of the unambiguous ground truth of the drift occurrence moments

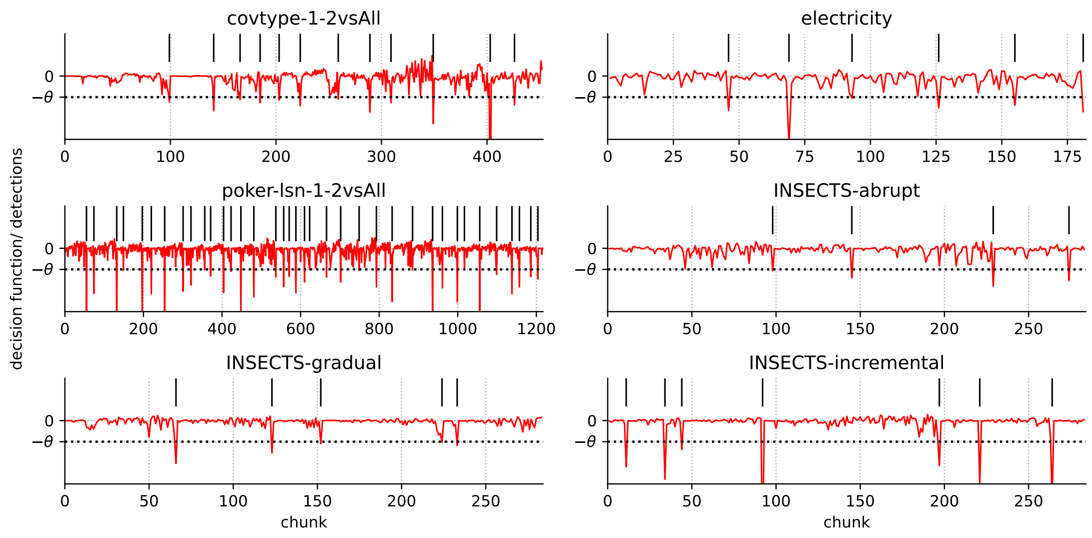

Figure 3.7: *The detections identified by* C2D *method for evaluated real-world data streams. The red line shows the mean value of the decision function used in the detection process, while the black vertical lines show the moments of detected concept change.*

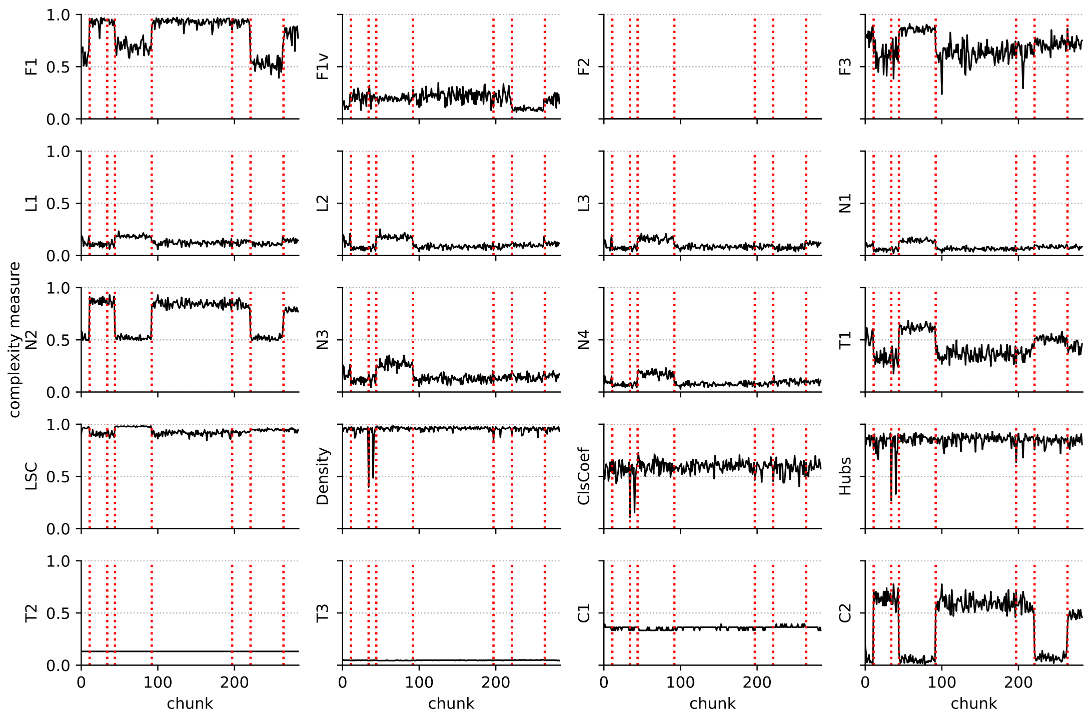

Figure 3.8: *Complexity measures during the processing of* INSECTS-incremental *data stream. The black lines indicate the metafeature values and the red vertical lines show the recognized detection moments.*

in real-world data streams. However, based on the average value of the decision function of a one-class classifier, it is possible to subjectively assess the stream variability and the validity of the detection. It can be seen that the deviations in the decision function vary during the flow. This phenomenon can be noticed in the *covtype* stream, where the penultimate drift signaling was recognized with an extreme deviation in the value of the decision function, while the intensity of changes was not so evident in the other detections. Controlling the algorithm's *threshold* hyperparameter may allow the model to become resistant to minor deviations or to be more sensitive to changes in problem difficulty.

A more detailed analysis of the variability of the classification problem complexity metrics is presented in Figure 3.8. Presented are the values of all problem complexity measures taken into account during the analysis of the `INSECTS-incremental` stream. Red lines mark the moments of concept change detection by the proposed method.

It can be seen that the metrics vary widely during stream processing. The most reactive metrics include $F1$, $F3$, $N2$, $T1$, $ClsCoef$, and $C2$ imbalance metrics. It is worth noting that the criteria considered during detection are characterized by diversity – drifts can be signaled by an incomplete and varying set of complexity measures.

The overall results show that the proposed C2D *method can effectively detect concept changes in the processing of both synthetic and real-world data streams. The experiments on the synthetic data showed plausible results expressed in drift detection error measures. The real-world data analysis exhibited that the proposed method can detect drifts in the data streams with large dimensionality, such as covtype described by 54 features, additionally providing the possibility of their explanation.*

3.2 Statistical Drift Detection Ensemble

This section presents the *Statistical Drift Detection Ensemble* (SDDE) method. This drift detection approach expands the work by Micevska et al. [163], in which authors proposed the *Statistical Drift Detection Method* (SDDM).

The proposed extension uses an ensemble approach to calculate the statistical metafeatures on the problem subspace level. This is a response to an original approach suffering from the *curse of dimensionality* – as the probability density estimation became less accurate in high dimensional spaces [227]. Similarly to SDDM, the proposed drift detector uses the metafeatures dedicated to the quantitative analysis of concept drift in the data stream proposed and described by Webb et al. [230, 231], described in detail in Section 1.1.5 – namely *drift magnitude* (DM), *conditioned cardinal covariate drift* (CMCD) and *posterior drift* (PD).

The first version of the SDDE, similarly to SDDM [163], used the *posterior drift* (PD) measure. However, after conducting preliminary experiments, *posterior drift* metafeature

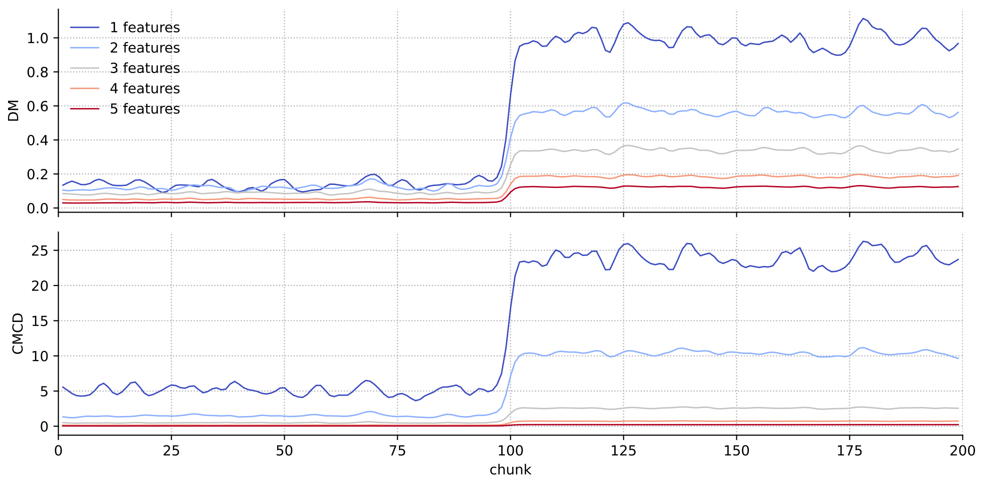

Figure 3.9: *The metafeatures describing drift magnitude (top row) and conditioned cardinal covariate drift (bottom row) calculated for the data stream with a sudden concept change present at* 100*th* *data chunk. The colors of plots describe the dimensionality of the data used to estimate the probability distribution. Obtained metafeature values were filtered with a Gaussian filter.*

was dropped from the pool of used measures due to the significant number of unjustified concept drift detections. Therefore, two measures are ultimately used in the proposed drift detector.

All the above-mentioned statistical measures are calculated based on the probability density distributions estimated using *Kernel Density Estimation* (KDE). *Kernel Density Estimation* is sensitive to the number of problem dimensions, as a large number of features can lead to performance degradation due to the curse of dimensionality. This phenomena was illustrated in Figure 3.9, showing the *drift magnitude* and *conditioned cardinal covariate drift* measures calculated for various dimensionalities of the data stream, where the sudden concept drift was injected in the central point of the examined data stream. The figure additionally shows the potential of those metafeatures in the drift detection task, as the moment of concept change can be easily identified.

Since the measures best describe the changes in low dimensionalities, SDDE estimates the densities of the probability distributions in the problem subspace set $\mathcal{S} = \{s_1, \ldots, s_e\}$, where e denotes the subspace number, and, at the same time, the size of the estimator pool. As shown in the Algorithm 2, presenting the operation of SDDE method, the set of subspaces is initialized at the beginning of the data stream processing procedure (line 1). Each problem subspace is created using sampling with replacement, and the n_s hyperparameter defines their size.

The method keeps two sets of KDE models for each subspace. One model is trained on the current, k-th data chunk, denoted by kde_k, and the second one is built based on the chunk in which the concept drift was last detected (or on the first chunk of the stream), denoted by kde_b. The initialization of kde_k is presented in line 4 of the pseudocode and kde_b – in line 7. Those models are later used to estimate the probability of the data density (lines 9:10), which allows for the calculation of monitored metafeatures (line 11). The hyperparameter λ specifies the number of chunks that need to be processed since the beginning of the stream to begin the detection task. If the required number of chunks has been processed, the number of detections is calculated according to the threshold θ and previously calculated metafeatures for specific subspaces (line 13). The drift is signaled by comparing such number with the value calculated according to the sensitivity δ of an ensemble (lines 14:15). If a drift is detected, *base* models kde_b are replaced with current ones kde_k (line 16).

The pseudocode was simplified to use the abstract functions, which are described as follows:

- *prepare_ subspaces*$(\mathcal{DS}_k, n_s, e)$ – sampling (with replacement) a set of e subspaces, where each subspace contains a fixed number of problem features equal to the value

Algorithm 2 Pseudocode of the *Statistical Drift Detection Ensemble*

```
Input:
    $\mathcal{DS} = \{\mathcal{DS}_1, \mathcal{DS}_2, \dots, \mathcal{DS}_k\}$ – data stream
                                                        ▷ Hyperparmeters
    $\theta$ – detection threshold
    $\delta$ – sensitivity of the ensemble
    $e$ – number of subspaces and size of the pool
    $n_s$ – subspace size
    $\lambda$ – number of chunks skipped before the first detection
                                                        ▷ Parmeters and functions
    $\mathcal{S}$ – set of feature subspaces
    $\mathcal{DM}$ – calculate DM measure
    $\mathcal{CMCD}$ – calculate CMCD measure
    prepare_subspaces – prepare problem subspaces
    fit_kernel_density – fit KDE
    estimate_density – estimate density with KDE
    get_detections – determine number of detections
                                                        ▷ Prepare a set of feature subspaces
 1: $\mathcal{S} = prepare\_subspaces(\mathcal{DS}_1, n, e)$
 2: for each $\mathcal{DS}_k \in \mathcal{DS}$ do          ▷ Fit a set of KDE models for k-th data chunk
 3:     for each $s_i$ in $\mathcal{S}$ do
 4:         $kde_k \leftarrow fit\_kernel\_density(\mathcal{DS}_k, s_i)$
 5:     end for                                         ▷ Initialize a set of base KDE models
 6:     if $k = 1$ then
 7:         $kde_b \leftarrow kde_k$
 8:     end if                                          ▷ Estimate probability density
 9:     $p_k \leftarrow estimate\_density(\mathcal{DS}_k, kde_k)$
10:     $p_b \leftarrow estimate\_density(\mathcal{DS}_k, kde_b)$
                                                        ▷ Calculate statistical metrics
11:     $dm_k, cmcd_k \leftarrow \mathcal{DM}(p_k, p_b), \mathcal{CMCD}(p_k, p_b)$
12:     if $k > \lambda$ then                           ▷ Determine number of detections in ensemble
13:         $c \leftarrow get\_detections(dm_k, cmcd_k, \theta)$
14:         if $c \geq 2 \times e \times \delta$ then   ▷ Drift detection
15:             signalize drift
16:             $kde_b \leftarrow kde_k$
17:         end if
18:     end if
19: end for
```

of the n_s hyperparameter. The data chunk $\mathcal{DS}_k$ is used to get the dimensionality of the data.

- *fit_kernel_density*$(\mathcal{DS}_k, s_i)$ – builds a set of KDE models on subspace s_i.
- *estimate_density*$(\mathcal{DS}_k, kde)$ – estimates the density of the probability distributions in the DS_k data chunk for each s_i subspace, using a set of previously trained *kde*.
- *get_detections*$(dm, cmcd, \theta)$ – determines, based on the historical values of the statistical metrics, the number of base detectors that identified a concept drift in the *k*-th data chunk. The final decision to detect the drift occurrence is based on the current DM and CMCD values and the harmonic mean of the historical values over the base detectors. The θ defines the threshold of the harmonic mean differences to indicate a concept drift.

Breaking down the problem into subspaces allows SDDE to avoid issues related to the large dimensionality and allows obtaining an ensemble of detectors instead of a single one.

In this case, the actual number of detectors in a pool corresponds to the number of subspaces e multiplied by the number of statistical measures used, which for the proposed SDDE algorithm is equal to 2.

Since the method is following an ensemble approach to concept drift detection, the decision about drift occurrence must be considered on two levels: (a) each base detector and (b) the entire ensemble. In the case of base detectors, the decision is made based on the distance between the values of the DM_k and CMCD_k statistical measures for the k-th data chunk and the mean of the historical values harmonic mean – calculated over each of the base detectors for all previous data chunks – to the standard deviation of the harmonic mean, based on the three-sigma rule. At the level of the entire ensemble, the simple comparison is made between the number of base detections and a value calculated according to the number of metafeatures, number of detectors e, and a sensitivity of the method δ.

In summary, the SDDE method builds a fixed-size pool of detectors, basing the recognition on the distribution density function of consecutive processing chunks. When the detection threshold is exceeded, i.e., a significant change between reference (kde_b) and current distribution (kde_k) is recognized by the ensemble, the estimated reference distributions are exchanged in each of its members, which allows updating the knowledge about the current concept. The critical element here is the batch exchange of base detectors, which are not updated at the time of individual recognition of a concept change but only after reaching a consensus within the ensemble – i.e., the drift detection by an entire system.

A simplified example of the SDDE operation is presented in Figure 3.10. It shows processing on a flow of thirteen chunks ($\mathcal{DS}_1$–$\mathcal{DS}_{13}$), where the ensemble is built on $e = 6$

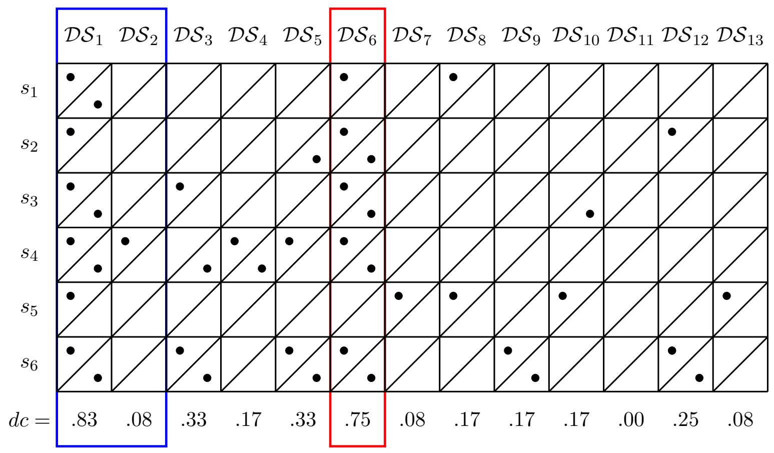
$\mathcal{DS}_1$ $\mathcal{DS}_2$ $\mathcal{DS}_3$ $\mathcal{DS}_4$ $\mathcal{DS}_5$ $\mathcal{DS}_6$ $\mathcal{DS}_7$ $\mathcal{DS}_8$ $\mathcal{DS}_9$ $\mathcal{DS}_{10}$ $\mathcal{DS}_{11}$ $\mathcal{DS}_{12}$ $\mathcal{DS}_{13}$
s_1
s_2
s_3
s_4
s_5
s_6
$dc =$ .83 .08 .33 .17 .33 .75 .08 .17 .17 .17 .00 .25 .08

Figure 3.10: *The example of* SDDE *processing in the first 13 data chunks of the data stream. The detection is skipped in the first two chunks, according to the λ hyperparameter, which is indicated by the blue area. The chunk shown in red is the only one in which the drift detection was signaled.*

random subspaces (s_1–s_6), assuming a λ hyperparameter of 2 and a sensitivity $\delta = 50\%$. Each illustration cell represents the response of a pair of detectors – DM and CMCD metafeatures – built on the same subspace (s_k).

Ten partial detections (dots) can be observed in the first chunk of processing, which is 83% of all detectors. However, this does not lead to drift detection as the model is not in the detection mode (blue area) – since the required number of λ chunks was not yet processed. In chunks $k = 3$ and $k = 5$, the drift is signaled by four detectors. However, it is only 33% of the available pool, so the ensemble is not yet reporting the drift, and each detector remains in its original state. In the $k = 6$ chunk, the 9 detections are signaled, constituting 75% of a possible number of detections and exceeding the *delta*. This leads to the recognition of a concept drift, marked in red.

3.2.1 The design of experiments

This subsection describes the data streams used during experiments, the setup of the conducted research, and the goals of the specific experiments performed on the SDDE method. Two main experiments were conducted – focusing on analyzing the method hyperparameters and the comparison with *state-of-the-art* drift detectors.

Data streams Experiments were carried out on synthetic data streams with various characteristics. To compare the proposed approach with reference methods, a vast range of data streams were generated, characterized by three types of drifts in the context of dynamics (*sudden*, *incremental*, and *gradual*), and with a different number of drifts (three, five, and seven drifts).

Table 3.8: *Generator configuration for synthetic data streams used in the experiments for the* SDDE *method. The first column describes the data characteristics, and the second one their specified values.*

CHARACTERISTICS	CONFIGURATION
Number of chunks	200
Chunk size	250
Number of features	10, 15, 20 (100% informative)
Drift frequency	3, 5, 7 drifts
Drift dynamics	*sudden*, *incremental*, *gradual*
Replications	10

Additionally, various dimensionalities were considered – 10, 15, and 20 informative features. Each of the generated streams consisted of 200 chunks of 250 objects. Finally, each stream with the given configuration was generated ten times to enable reliable experimental evaluation. The full description of generated data streams is presented in Table 3.8.

A reduced number of data streams was evaluated in the first experiment, which concerned the selection of the method's hyperparameters. Optimization was performed for streams consisting of 100 chunks of 200 objects, each described by 15 features. Three types of drift dynamics were examined. Ten streams were generated for each configuration. Both experiments used the three *drift detection error* measures as the primary evaluation criteria.

Hyperparameter optimization The first experiment aimed to optimize the SDDE method's hyperparameters, such as *sensitivity*, *subspace size*, and the *number of detectors.*

The following hyperparameter values were considered:

- *subspace size* (n_s) – 1, 2, or 3 random features from the initial feature space.

  Research in [163] has shown that *drift magnitude* measures provide the most information for low dimensional spaces. Moreover, the underlying *Kernel Density Estimators* are prone to the curse of dimensionality. This motivated the selection of low *subspace size* values for evaluation.

- *detector's sensitivity* (δ) – 20 values uniformly sampled form 0% to 100%.

  The *sensitivity* hyperparameter specifies the fraction of detectors in the ensemble that must detect drift for it to be considered as an integrated, final decision. The hyperparameter set to zero is the equivalent of stable, deterministic drift detection on every data chunk, even when none of the detectors indicates it. Likewise, a 100% value requires all detectors to be acclaimed, making the most strict integration rule. Examining the full range of hyperparameter values allows for a detailed analysis of its effect on an algorithm.

- *number of detectors* (e) – 20 evaluated values ranging from 1 detector to 100, sampled from quadratic function.

  Increasing the *number of detectors* may potentially increase detection quality but may also increase memory and computational complexity. For the evaluated streams containing 15 features, increasing the number of detectors increases the chance of analyzing all features during processing.

Comparison with the reference methods The second experiment aimed to compare the proposed SDDE method with reference drift detection algorithms. For this purpose, five methods presented in Table 3.9 were considered. The selected reference approaches are among the canonical, *state-of-the-art* drift detectors. Their hyperparameters were selected according to the default values from *scikit-multiflow* [170] library.

Table 3.9: *The configuration of reference drift detectors for comparison experiment with* SDDE *approach. The first two columns specify the method acronym and its name, and the final column – its configuration.*

METHOD		CONFIGURATION
DDM	*Drift Detection Method* [70]	default detection *threshold* of 3, the base classifier used for error monitoring was Gaussian Naive Bayes; the *skip* hyperparameter was set to 30
EDDM	*Early Drift Detection Method* [12]	default *beta* of 0.9, the base classifier used for error monitoring was Gaussian Naive Bayes
ADWIN	*Adaptive Windowing* [24]	default *delta* of 0.002, the base classifier used for error monitoring was Gaussian Naive Bayes
HDDM$_A$	*Hoeffding Drift Detection Method* with *Bounding Moving Averages* [66]	the default *drift level* of 0.001, the base classifier used for error monitoring was Gaussian Naive Bayes
HDDM$_W$	*Hoeffding Drift Detection Method* with *Bounding Weighted Moving Averages* [66]	the default *drift level* of 0.001 and λ of 0.05, the base classifier used for error monitoring was Gaussian Naive Bayes

The proposed SDDE method was hyperparameterized by *sensitivity*, *number of detectors*, and *subspace size* selected during the analysis of the first experiment. The *sensitivity* hyperparameter was calibrated for each type of drift dynamics.

3.2.2 Experimental evaluation

This section presents the results and critical analysis of the conducted experimental evaluation, which consisted of experiments searching for the optimal hyperparameterization of the SDDE method and reviewing its effectiveness compared to *state-of-the-art* methods.

Hyperparameter optimization The first experiment aimed to select the values of the SDDE hyperparameters for further processing. Figure 3.11 shows the results for a single data stream type and subspace size $n_s = 1$ to present the specific criteria and their combination. For the selected *subspace size*, it presents the heatmaps describing the dependence of the three assessment criteria ($D1$, $D2$, and R) in the space of the ensemble size e, and the sensitivity δ of ensemble's decision integration.

The figure shows three drift detection error measures: $D1$ – the average distance from each detection to the nearest drift, $D2$ – the average distance of each drift to the nearest detection, and R – the adjusted ratio of the number of drifts to the number of detections. It is worth mentioning again that the single criterion does not allow for reliable drift detection evaluation. Hence, a combination of all three errors is necessary to correctly assess the method operation.

These factors are normalized and aggregated in the final subplot in Figure 3.11. The image presents a color combination of RGB channels defined by the base metrics – $D1$

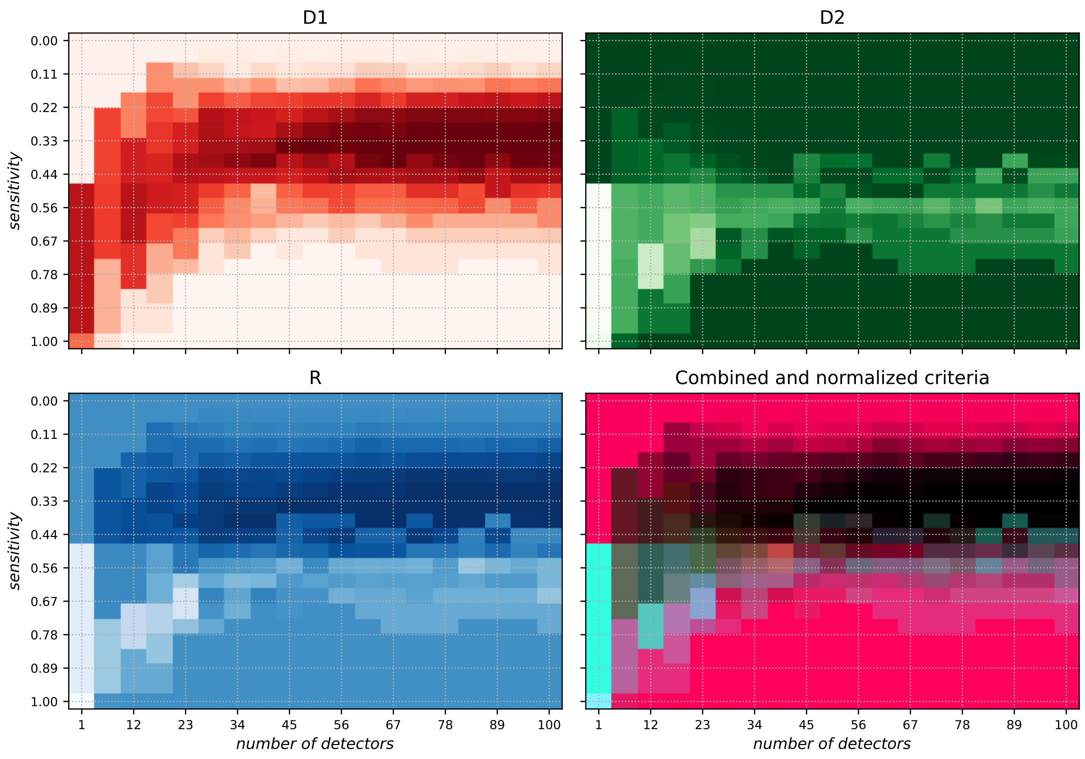

Figure 3.11: *The illustrative result of the first experiment for a single data stream configuration – for sudden concept drift and subspace size of $n_s = 1$. The results show drift detection errors dependent on the number of detectors in ensemble e (horizontal axis) and sensitivity δ (vertical axis).*

in the red color channel, $D2$ in the green color channel, and R in blue. Bright cells represent the configuration with the globally highest errors, while the most saturated colors of magenta and cyan show the worst configurations according to the $D1$ and $D2$ criteria, respectively. The dark area in the illustration represents the region of the most promising configurations. Responses from the drift detection error measures confirm the observations about the detection stability after reaching a certain number of detectors e.

Figure 3.12 shows the color combination for drift detection error measures for three examined drift dynamics (in rows) and for the three *subspace size* values (in columns). Similarly, the darkest areas describe the best hyperparametrization of the method. In gradual and incremental drifts, the darkest area is less visible, indicating that the errors for this type of change are characterized by a more significant deviation. There is a small range of hyperparameters for which all three drift detection error measures are low.

After reaching a sufficiently large number of detectors, the stabilization of measurements is clearly noticeable for all drift types. The most stable in this context, as previously noticed in work presenting SDDM [163], seems to be an independent analysis of the features. The default *number of detectors* e in an ensemble was set to 20 – which is a compromise between the quality of method operation and the computational complexity related

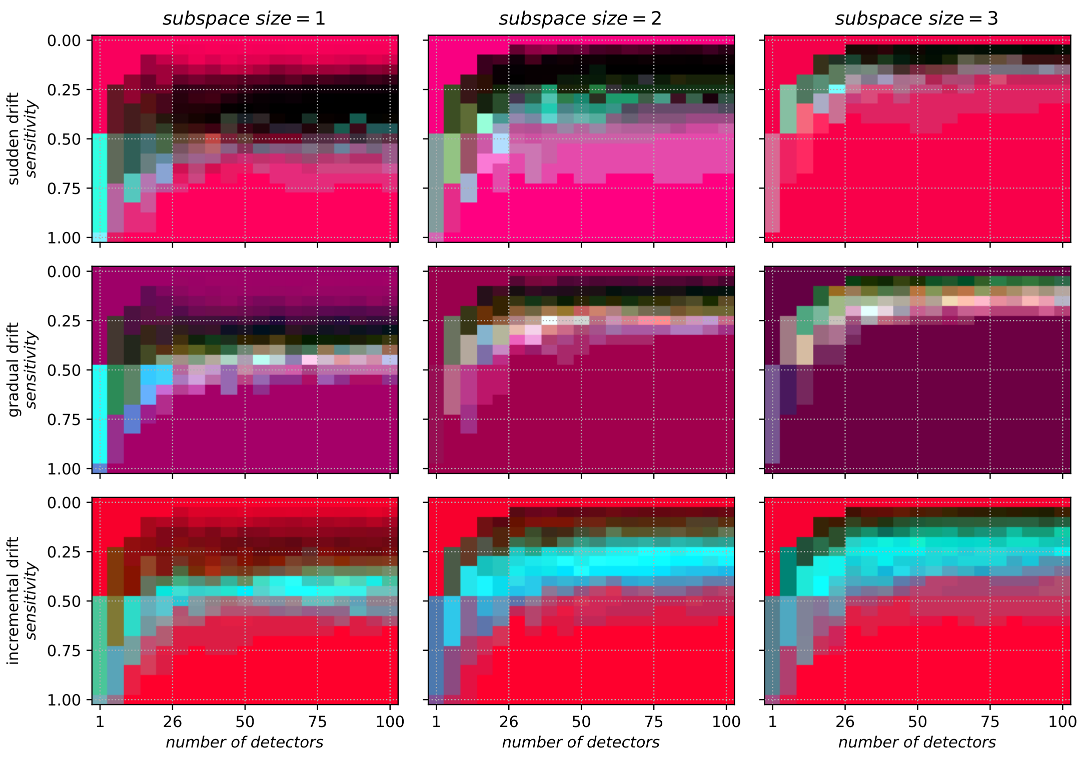

Figure 3.12: *The complete results for the second experiment – for types of drift dynamics presented in rows of the figure, and the subspace size (n_s) hyperparameter in columns. The heatmaps present the normalized and combined drift detection error measures.*

to the need for numerous parallel drift detections. The *sensitivity* δ of the method was selected depending on the drift dynamics – 45% for sudden drifts, 40% for gradual, and 35% for incremental changes.

Comparison with the reference methods The second experiment aimed to compare the operation of the proposed approach to the *state-of-the-art* drift detection methods. Figures 3.13–3.15 show the drift detection moments for streams with sudden, gradual, and incremental drift dynamics, respectively. The rows present the detections made by evaluated methods in ten stream replications. The detections made by the proposed approach are marked in red, and the last row presents the drift dynamics. In each figure, the columns present the results for different stream dimensionalities, and the rows for different numbers of concept changes present over the stream course.

The first visible observation in the case of this analysis is the apparent hyperactivity of the EDDM detector, leading to redundant alerts on the entire course of the streams. There is a decrease in their density at the moments of stable concepts (evident with sudden drift), but it is still a method that indicates a change of concept extremely often. The DDM method behaves similarly, especially in the initial phase of streams. In many

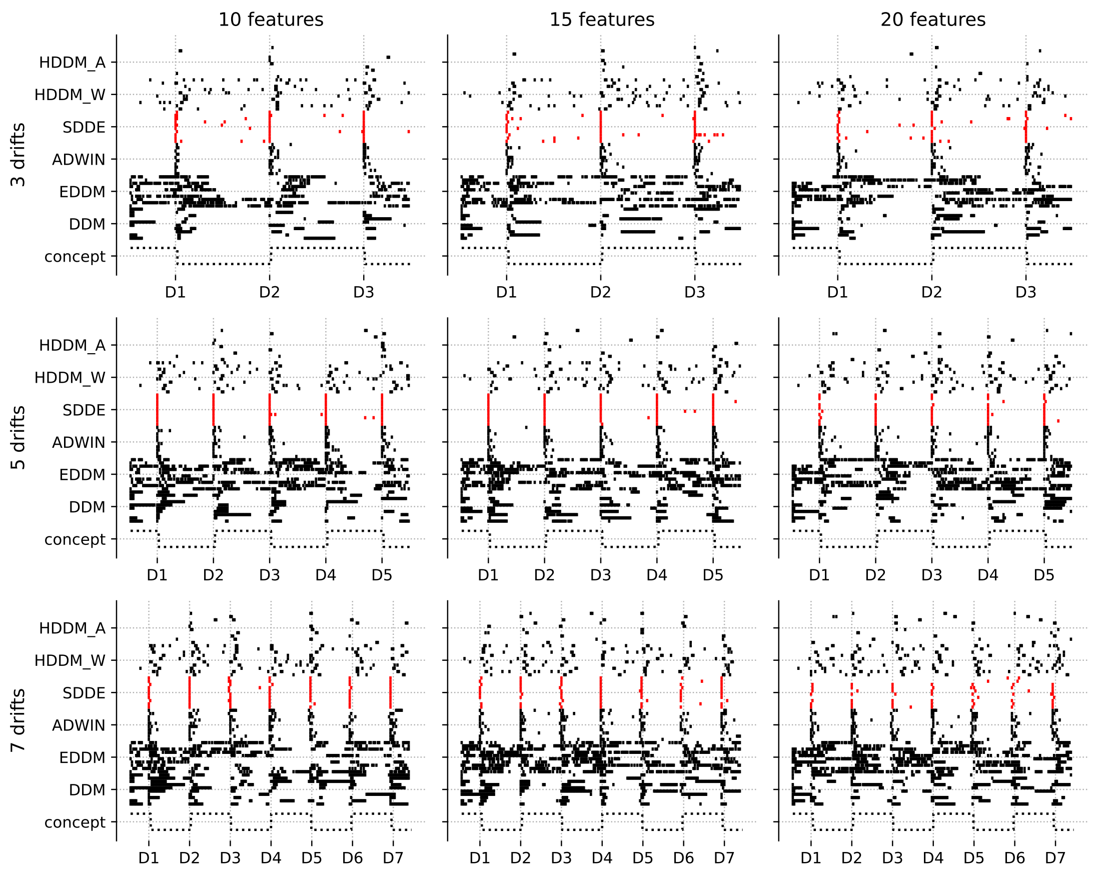

Figure 3.13: *The results of the comparison experiment for* SDDE *evaluation for streams with* sudden *concept change. The points indicate the moments of concept drift detection, with the proposed method highlighted in red.*

cases, it either quickly becomes desensitized to changes, ceasing to be useful for concept drift recognition, or leads to continuous alerts that begin with the emergence of a new concept and continue throughout its processing.

The results for the ADWIN and SDDE methods present much better – in the case of sudden drifts, the detections often cover the line of occurrence of the drift almost perfectly. In the case of gradual drifts, they also behave quite similarly – however – they spread the detection points wider over the entire course of the ongoing change. It is sometimes alerting several times throughout a single change, highlighting the transition phases of the drift and, at the same time, allowing for detection adequately earlier than the central drift point.

Apparent differences between SDDE and ADWIN appear only in the case of incremental drifts, where the standard deviation of the detection distance from the drift is more significant for SDDE while maintaining a uniform distribution around the central drift point, which can be interpreted as a higher ability of the proposition presented in this work to signal a drift early.

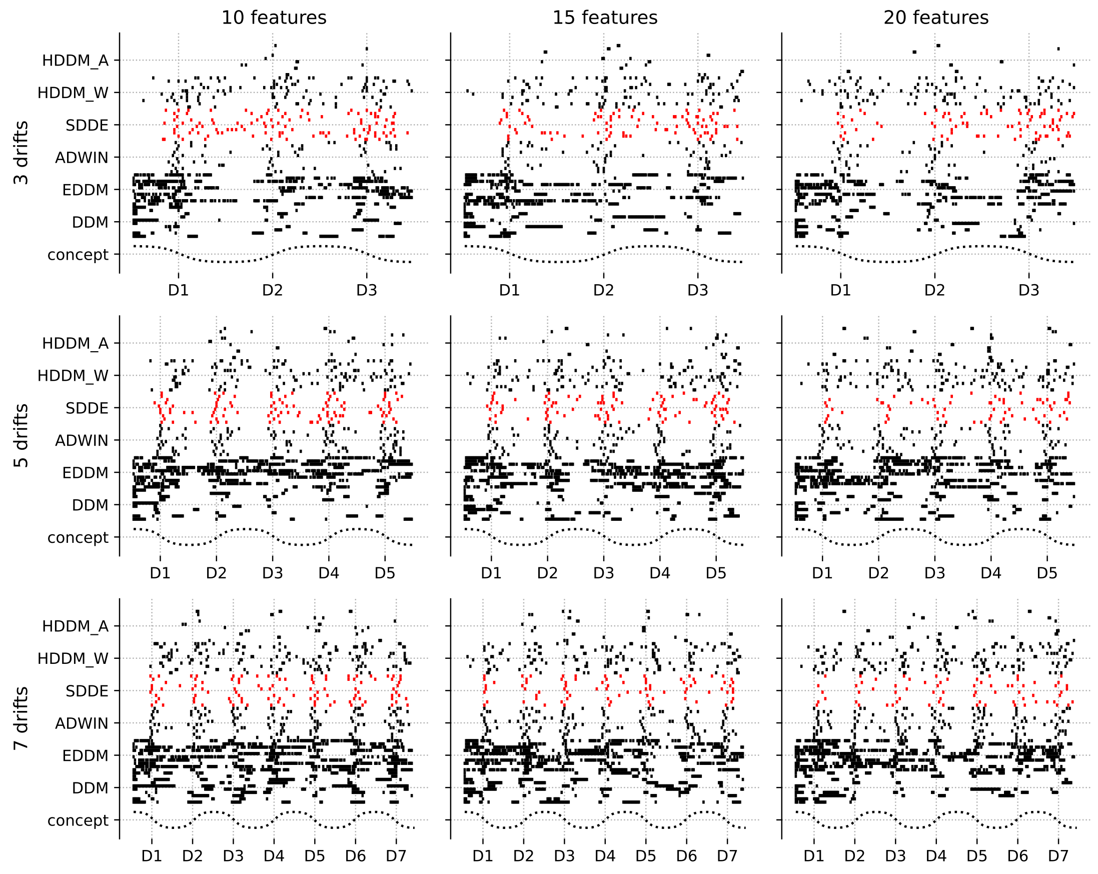

Figure 3.14: *The results of the comparison experiment for* SDDE *evaluation for streams with* gradual *concept change. The points indicate the moments of concept drift detection, with the proposed method highlighted in red.*

The HDDM$_A$ method recognizes drifts only in a few replications, especially in the case of streams characterized by rare drifts. Generally, the method presents low sensitivity to concept drifts. However, the HDDM$_W$ detector shows frequent detections, which rarely lay in the exact moment of drift but are instead signaled after a certain number of chunks describing a new concept.

Tables 3.10 – 3.12 present the exact results expressed in three drift detection error measures for the performed comparison experiment. The column presenting the proposed SDDE method is marked in red. The table also contains the statistical Student's T-test results performed for a significance level of $alpha = 5\%$. Below each average error value presented in the table, the indexes of the methods from which the given one is statistically significantly better are presented. For each stream type presented in rows, the method that was statistically significantly better than the largest number of reference methods is emphasized in bold.

Table 3.10 shows the results for the $D1$ measure examining the average distance from each detection to the nearest drift. It can be seen that the worst results in this metric

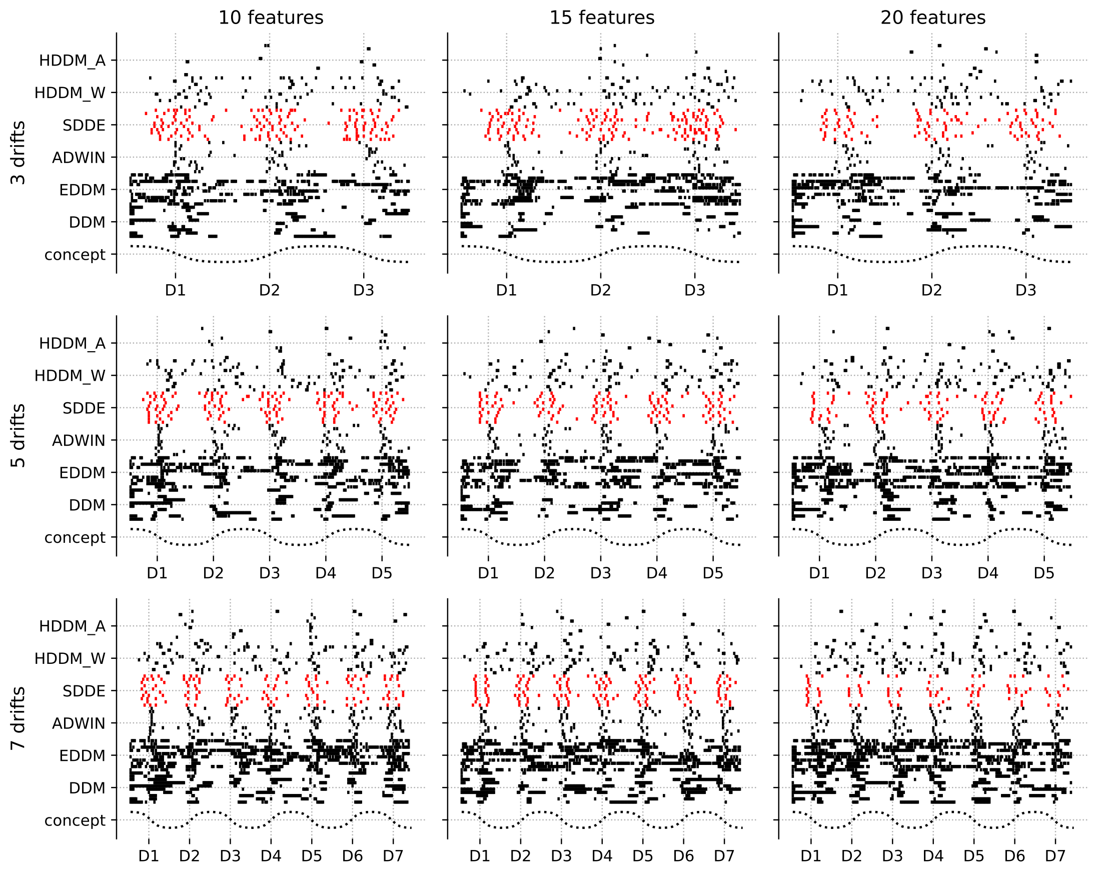

Figure 3.15: *The results of the comparison experiment for* SDDE *evaluation for streams with* incremental *concept change. The points indicate the moments of concept drift detection, with the proposed method highlighted in red.*

are obtained by the DDM and EDDM methods, which are characterized by numerous redundant detections, causing an increase in the average distance from detection to drift. The HDDM$_W$ method obtained low results due to the large spread of drift signaling, while the HDDM$_A$ method does not perform well in this measure due to the lack of detection signaling for part of the stream replication, which meant calculating the metric according to the assumption of detection signaling for all chunks of the stream. The ADWIN and SDDE methods perform best within this error measure.

Table 3.11 shows the results for the $D2$ measure examining the average distance from each drift to the nearest detection. In this criterion, the worst results are obtained by the HDDM$_A$ method, in which very rarely reported detections were distant from the drifts actually occurring in the stream. Similarly, for the tested streams, poor results are obtained by the DDM and HDDM$_W$ methods, which statistically outperform only the worst results of HDDM$_A$. Frequent reports of drift in the EDDM method translate into its high quality within this metric, which is evident in gradual and incremental drifts. Redundant detections made by the method do not harm the values of this metric, as only

Table 3.10: *Results of comparison experiment in the D1 drift detection error measure. The columns present the evaluated methods, with the proposed method highlighted in red, and the rows show the type of data stream in the evaluation.*

			DETECTION FROM NEAREST DRIFT (D1)					
			DDM (1)	*EDDM* (2)	*ADWIN* (3)	*SDDE* (4)	$HDDM_W$ (5)	$HDDM_A$ (6)
SUDDEN	3 DRIFTS	10F	14.346	16.130	**1.958**	**4.443**	10.604	**5.175**
			—	—	**1 2 5**	**1 2 5**	2	**1 2 5**
		15F	18.602	16.919	**2.352**	**4.320**	12.282	**4.633**
			—	—	**1 2 5**	**1 2 5**	1 2	**1 2 5**
		20F	113.684	17.106	**2.473**	4.905	9.674	11.925
			2	—	**all**	1 2 5 6	1 2	—
	5 DRIFTS	10F	8.962	9.942	2.003	**0.357**	7.576	4.875
			—	—	1 2 5	**all**	2	1 2
		15F	8.333	9.513	**1.487**	**0.858**	6.895	6.367
			—	—	**1 2 5 6**	**1 2 5 6**	1 2	—
		20F	8.998	10.641	**2.179**	**0.897**	7.268	7.310
			2	—	**1 2 5 6**	**1 2 5 6**	1 2	—
	7 DRIFTS	10F	5.860	6.649	1.623	**1.020**	4.990	2.897
			—	—	1 2 5	**all**	2	1 2 5
		15F	4.965	6.850	2.084	**1.337**	5.186	2.693
			2	—	1 2 5	**1 2 3 5**	2	1 2 5
		20F	5.793	6.925	**1.697**	**2.164**	5.984	5.741
			2	—	**1 2 5 6**	**1 2 5 6**	2	—
GRADUAL	3 DRIFTS	10F	13.552	14.304	**10.074**	**11.871**	13.076	9.933
			—	—	**2**	**2**	—	—
		15F	18.354	14.319	**8.367**	11.500	14.328	16.200
			—	—	**1 2 5**	1 5	—	—
		20F	15.285	14.037	**7.832**	13.460	13.461	15.266
			—	—	**all**	—	—	—
	5 DRIFTS	10F	7.994	8.872	**5.789**	**4.974**	9.094	8.467
			—	—	**2 5 6**	**2 5 6**	—	—
		15F	8.480	8.743	**6.051**	**5.353**	9.139	6.800
			—	—	**1 2 5**	**1 2 5**	—	—
		20F	9.070	8.726	**6.245**	7.896	8.748	6.887
			—	—	**1 2 5**	—	—	—
	7 DRIFTS	10f	5.792	6.250	**3.718**	**4.033**	6.582	4.688
			—	—	**1 2 5**	**1 2 5**	—	—
		15F	6.604	6.275	**4.048**	**3.769**	6.926	**3.592**
			—	—	**1 2 5**	**1 2 5**	—	**1 2 5**
		20F	6.589	6.488	**4.148**	4.945	6.378	6.400
			—	—	**1 2 5 6**	1 2 5	—	—
INCREMENTAL	3 DRIFTS	10F	15.591	15.655	**9.718**	**9.703**	14.315	11.425
			—	—	**1 2 5**	**1 2 5**	—	—
		15F	18.384	15.826	**8.621**	**10.228**	15.499	10.250
			—	—	**1 2 5**	**1 2 5**	—	—
		20F	16.592	17.698	**9.224**	**9.837**	15.285	18.075
			—	—	**1 2 5 6**	**1 2 5 6**	—	—
	5 DRIFTS	10F	9.193	8.627	**5.380**	**5.507**	9.974	**4.880**
			—	5	**1 2 5**	**1 2 5**	—	**1 2 5**
		15F	9.027	9.694	**4.669**	5.789	9.302	7.283
			—	—	**1 2 4 5**	1 2 5	—	—
		20F	9.206	9.336	**5.327**	**5.280**	10.047	9.062
			—	—	**1 2 5 6**	**1 2 5 6**	—	—
	7 DRIFTS	10F	6.376	6.376	**3.903**	**4.223**	7.284	3.824
			—	—	**1 2 5**	**1 2 5**	—	1 5
		15F	5.894	6.546	**3.691**	**3.663**	7.425	4.700
			—	5	**1 2 5**	**1 2 5**	—	5
		20F	66.442	6.742	**3.540**	**4.093**	7.230	6.833
			—	—	**1 2 5 6**	**1 2 5 6**	—	—

the single detection closest to the drift is considered during evaluation. For streams with sudden drifts or only a few changes (three concept drifts), ADWIN and SDDE detectors perform best. It is worth highlighting the case of streams with five sudden drifts in 10-dimensional stream, in which the SDDE method signaled the drift precisely at the moment of its occurrence for all stream replications, which resulted in an error measure $D2$ of 0.

Again, in the R measure, shown in Table 3.12, as in the case of the $D1$ measure, DDM and EDDM methods are penalized for too many detections. This is evidenced by high

Table 3.11: *Results of comparison experiment in the D2 drift detection error measure. The columns present the evaluated methods, with the proposed method highlighted in red, and the rows show the type of data stream in the evaluation.*

			DRIFT FROM NEAREST DETECTION (D2)					
			DDM (1)	*EDDM* (2)	*ADWIN* (3)	*SDDE* (4)	$HDDM_W$ (5)	$HDDM_A$ (6)
SUDDEN	3 DRIFTS	10F	16.333	6.000	0.667	**0.067**	4.633	43.967
			6	6	1 2 5 6	**all**	6	—
		15F	20.100	5.267	**0.367**	**0.200**	5.433	51.500
			6	6	**2 5 6**	**2 5 6**	6	—
		20F	8.133	4.867	**0.400**	**0.267**	4.900	36.333
			6	6	**1 2 5 6**	**1 2 5 6**	6	—
	5 DRIFTS	10F	6.880	3.280	0.400	**0.000**	4.520	39.340
			6	6	1 2 5 6	**all**	6	—
		15F	6.600	2.260	0.140	**0.020**	4.680	35.500
			6	5 6	2 5 6	**2 3 5 6**	6	—
		20F	8.500	2.460	**0.220**	5.780	4.780	35.360
			6	5 6	**1 2 5 6**	6	6	—
	7 DRIFTS	10F	5.186	11.700	**1.786**	**0.929**	4.000	27.400
			6	—	**5 6**	**5 6**	6	—
		15F	5.586	**1.114**	**1.586**	2.543	5.143	24.443
			6	**5 6**	**5 6**	6	6	—
		20F	12.786	**1.971**	**1.029**	17.057	4.314	20.271
			—	**5 6**	**5 6**	—	6	—
GRADUAL	3 DRIFTS	10F	11.567	6.633	8.467	**3.033**	5.033	37.300
			6	6	6	**1 6**	6	—
		15F	38.167	**5.967**	**3.367**	**2.933**	**7.167**	55.233
			—	**1 6**	**1 6**	**1 6**	**1 6**	—
		20F	10.467	**3.467**	**3.767**	4.533	5.633	30.600
			—	**1 6**	**1 6**	6	6	—
	5 DRIFTS	10F	21.740	2.480	3.960	**1.580**	5.900	41.280
			—	5 6	6	**1 5 6**	6	—
		15F	16.720	**1.400**	2.400	1.940	6.300	38.140
			—	**3 5 6**	5 6	5 6	6	—
		20F	21.360	**1.420**	1.860	11.000	4.680	30.560
			—	**4 5 6**	5 6	6	6	—
	7 DRIFTS	10f	6.657	**1.486**	**2.686**	3.857	5.671	30.271
			6	**5 6**	**5 6**	6	6	—
		15F	11.386	**2.529**	2.171	5.343	5.914	33.986
			6	**1 5 6**	5 6	6	6	—
		20F	28.200	**1.557**	**1.843**	11.886	4.986	28.571
			—	**1 4 5 6**	**1 4 5 6**	6	1 6	—
INCREMENTAL	3 DRIFTS	10F	12.467	7.467	3.500	**1.767**	9.133	37.667
			6	1 6	1 2 5 6	**all**	6	—
		15F	20.133	**4.033**	**2.733**	**2.600**	8.833	59.000
			6	**5 6**	**5 6**	**5 6**	6	—
		20F	19.967	**5.467**	**4.267**	**3.000**	8.933	49.067
			6	**1 5 6**	**1 5 6**	**1 5 6**	6	—
	5 DRIFTS	10F	7.800	**3.120**	3.960	**2.080**	7.560	43.700
			6	**1 5 6**	5 6	**1 5 6**	6	—
		15F	19.500	**2.100**	2.320	2.760	6.800	42.180
			6	**1 5 6**	5 6	5 6	6	—
		20F	7.820	**1.640**	2.540	8.300	7.800	40.500
			6	**1 3 5 6**	1 5 6	6	6	—
	7 DRIFTS	10F	6.214	**1.386**	**1.871**	4.414	6.157	25.671
			6	**1 4 5 6**	**1 4 5 6**	6	6	—
		15F	6.557	**1.543**	**1.943**	2.500	7.500	30.186
			6	**1 5 6**	**1 5 6**	5 6	6	—
		20F	4.129	**0.886**	1.929	17.200	6.857	28.700
			5 6	**1 3 5 6**	1 5 6	—	6	—

error values close to (but not reaching) the value of one. On the other hand, the HDDM$_A$ method obtains error values exceeding one, which indicates that the number of detections is too small in relation to the drifts occurring in the stream. In the case of sudden drifts, the SDDE method achieves the best results, often statistically significantly better than all other approaches. Regarding gradual and incremental changes, the ADWIN detector achieves equally high results, particularly for incremental streams, being statistically significantly better than the proposed solution in six out of nine cases. This results from

Table 3.12: *Results of comparison experiment in the R drift detection error measure. The columns present the evaluated methods, with the proposed method highlighted in red, and the rows show the type of data stream in the evaluation.*

			RATIO OF DRIFTS TO DETECTIONS (R)					
			DDM (1)	*EDDM* (2)	*ADWIN* (3)	*SDDE* (4)	*HDDM$_W$* (5)	*HDDM$_A$* (6)
SUDDEN	3 DRIFTS	10F	0.833	0.895	0.495	**0.302**	0.662	0.948
			—	—	1 2 5	**all**	1 2	—
		15F	0.965	0.927	**0.507**	**0.280**	0.695	0.800
			—	—	**1 2 5**	**1 2 5**	2	—
		20F	0.768	0.941	**0.494**	**0.310**	0.651	1.095
			2	—	**1 2 5 6**	**1 2 5 6**	2	—
	5 DRIFTS	10F	0.827	0.908	0.513	**0.066**	0.601	1.083
			2	—	1 2	**all**	1 2	—
		15F	0.822	0.921	0.490	**0.062**	0.578	0.750
			2	—	1 2	**all**	1 2	—
		20F	0.816	0.924	0.513	**0.183**	0.601	0.831
			2	—	1 2 5	**all**	1 2	—
	7 DRIFTS	10F	0.832	0.839	0.523	**0.037**	0.547	0.972
			—	—	1 2	**all**	1 2	—
		15F	0.754	0.909	0.486	**0.052**	0.504	0.762
			2	—	1 2	**all**	1 2	—
		20F	0.766	0.903	**0.497**	0.787	**0.556**	0.674
			2	—	**1 2**	—	**1 2**	—
GRADUAL	3 DRIFTS	10F	0.782	0.928	**0.523**	0.716	0.702	1.147
			2	—	**all**	2	2	—
		15F	0.911	0.906	**0.496**	0.647	0.707	**0.350**
			—	—	**1 2 4 5**	2	2	**1 2 4 5**
		20F	0.781	0.910	**0.440**	**0.576**	0.719	0.769
			2	—	**1 2 5**	**1 2 5**	2	—
	5 DRIFTS	10F	0.667	0.920	0.525	**0.468**	0.595	1.000
			2	—	2 6	**2 5 6**	2	—
		15F	0.755	0.917	**0.457**	**0.439**	0.613	1.208
			2	—	**1 2 5**	**1 2 5**	1 2	—
		20F	0.761	0.917	**0.444**	**0.390**	0.630	0.814
			2	—	**1 2 5**	**1 2 5**	1 2	—
	7 DRIFTS	10f	0.776	0.898	0.409	**0.344**	0.530	1.357
			2	—	1 2 6	**1 2 5 6**	1 2 6	—
		15F	0.718	0.871	**0.395**	**0.257**	0.510	1.765
			2	—	**1 2 5 6**	**1 2 5 6**	1 2 6	—
		20F	0.588	0.892	**0.421**	0.529	0.560	0.935
			2	—	**2 5**	2	2	—
INCREMENTAL	3 DRIFTS	10F	0.847	0.917	**0.536**	0.775	**0.620**	0.620
			—	—	**1 2 4**	2	**1 2 4**	1 2
		15F	0.950	0.941	**0.489**	0.797	0.678	1.550
			6	6	**all**	2 6	2 4 6	—
		20F	0.781	0.934	**0.439**	0.691	0.615	0.648
			2	—	**1 2 4 5**	2	2	—
	5 DRIFTS	10F	0.820	0.909	**0.422**	0.632	0.569	1.267
			2	—	**all**	1 2	1 2	—
		15F	0.835	0.919	**0.393**	0.610	0.570	1.625
			—	—	**all**	1 2	1 2	—
		20F	0.813	0.927	**0.404**	0.905	0.577	0.989
			2	—	**1 2 5 6**	—	1 2 6	—
	7 DRIFTS	10F	0.821	0.907	**0.449**	**0.381**	**0.502**	1.398
			2	—	**1 2**	**1 2**	**1 2**	—
		15F	0.753	0.904	**0.368**	0.502	0.500	1.333
			2 6	—	**all**	1 2 6	1 2 6	—
		20F	0.833	0.918	**0.361**	1.152	0.491	0.867
			2	—	**1 2 5**	—	1 2	—

the repeated signaling of each drift when it is stretched over a longer period (gradual and incremental changes) by the SDDE method. As already noted in the case of the C2D detector, described in Section 3.1, multiple signaling of a long-term change may bring benefits resulting from repeated adaptation of the underlying recognition method to changes extended over time. This behavior of the method can be used as a benefit, however, it negatively affects the value of the R measure, according to which each concept drift should be marked once.

Considering all of the obtained results, the proposed SDDE *detector achieves satisfactory drift detection error values for the evaluated data streams. The proposed ensemble approach allowed for the analysis of the monitored measures over the extended dimensionality of the processed data. The detector is particularly effective in detecting sudden drifts and is highly sensitive to gradual and incremental changes.*

3.3 Parallel Activations Drift Detector

This section presents an unsupervised *Parallel Activations Drift Detector* (PADD). A proposed method monitors the activations of a randomly initialized neural network (NN), addressing an important issue of label availability in the data streams – where the provided label can arrive with a time delay or – taking into account the cost of object annotation [86, 152] – may entirely not be possible to obtain.

The proposed drift detector uses a neural network whose weights are not updated during the stream processing. The outputs of such NN describe a set of random projections connected to a set of *activation functions* between individual layers. Such an order of transformations is deterministic, assuming no weight optimization. After aggregating the outputs of NN for samples within a data batch – for example, using an averaging function – an obtained value can be considered a *metafeature* of a given data chunk. However, such a metafeature does not have any straightforward semantics.

Depending on the number of random projections or the number of outputs from the neural network, a set of metafeatures describing a particular data portion can be obtained. As mentioned, those values have no semantic meaning, however, monitoring them could allow for noticing changes occurring in the data distribution without the use of sample's labels. The great advantage of such a set of metafeatures is the lack of the need to acquire labels to describe the data. Additionally, which is aligned with the research on neural network behavior [90], the approach should allow for the effective analysis of high-dimensional problem spaces. The complete procedure of the proposed PADD approach is described in the Algorithm 3.

The method processes data streams divided into non-interlacing batches ($\mathcal{DS}_k \in \mathcal{DS}$). Drift detection is marked based on t replications of statistical tests – aiming to validate the null hypothesis stating the lack of significant difference between two groups of independent measurements – comparing (a) a sample of size s from the distribution of past and (b) current activations at the all e outputs of the NN. The initial – and constant during the full processing – random weights of NN are drawn from a normal distribution. As default settings, the normal distribution has an expected value of zero and a standard

Algorithm 3 Pseudocode of the *Parallel Activations Drift Detection*

```
Input:
    $\mathcal{DS} = \{\mathcal{DS}_1, \mathcal{DS}_2, \dots, \mathcal{DS}_k\}$ – data stream
                                                              ▷ Hyperparameters
    $\alpha$ – significance level for statistical test
    $\theta$ – threshold for drift detection
    $t$ – number of statistical test replications performed for each NN output
    $s$ – number of samples drawn for statistical test
    $e$ – number of outputs from the NN
                                                              ▷ Parameters
    $\mathcal{C}$ – stored activation values for all NN outputs since last drift
    $\mathcal{NN}()$ – forward pass from NN model with random weights and $e$ outputs
    $\mathcal{T}$ – statistical test for sample subset comparison
    $a$ – counter of tests signaling statistically significant difference

 1: for all $\mathcal{DS}_k \in \mathcal{DS}$ do
 2:     $c \leftarrow \mathcal{NN}(\mathcal{DS}_k)$        ▷ Get ultimate network activations for current data chunk
 3:     if $\mathcal{C}$ is not empty then
 4:         $a \leftarrow 0$                               ▷ Initialize counter as zero
 5:         for all $e_i \in e$ do
 6:             for all $t_i \in t$ do
 7:                 $cc \leftarrow$ random $s$ from $c[e_i]$   ▷ Randomly select samples from current and past certainty
 8:                 $pc \leftarrow$ random $s$ from $C[e_i]$
 9:                 $p \leftarrow \mathcal{T}(pc, cc)$          ▷ Compute p-value of statistical test
10:                 if $p < \alpha$ then
11:                     increment $a$
12:                 end if
13:             end for
14:         end for
15:         if $a > \theta \times e \times t$ then          ▷ Drift detected
16:             $\mathcal{C} \leftarrow \emptyset$
17:             signalize drift
18:         end if
19:     end if
20:     store $c$ in $\mathcal{C}$                          ▷ Store current activations for future comparison
21: end for
```

deviation of 0.1. The statistical test used for distribution comparison is the unpaired Student's T-test, following the default selection of statistical comparison strategy for two independent, normally distributed groups describing a continuous variable [182].

The two most critical hyperparameters of the method are the significance level *alpha* (α) and the *threshold* hyperparameter (θ), indicating the fraction of all tests that need to signal statistical independence of distributions to raise a drift detection signal.

At the beginning of each batch processing, c activations are calculated for samples from a given batch at all NN outputs (line 2). In the first chunk, the historical activations $\mathcal{C}$ are yet unknown, so the detection step is skipped due to the lack of reference data. The current activations c are stored in the pool of historical outputs $\mathcal{C}$. Otherwise, statistical tests are performed for individual network outputs. In lines 5:14 of the pseudocode, t replications of the statistical test are performed for each e_i output of the network. Samples of size s are drawn with replacement from historical activations for a given output $\mathcal{C}$ and for the current distribution c. If the statistical test shows a significant difference between the past and current distribution, the counter a is incremented. The detection criterion is described in lines 15:18 of the pseudocode. If the counter a exceeds the required number of tests showing statistically significant (difference defined using θ, the number of outputs e, and the number of test replications t), a concept drift in the current chunk is signaled. Such a signal triggers clearing the buffer of past supports ($\mathcal{C}$). For each batch, the current activations c are saved to the historical data at the end of processing (line 20).

The Student's T-test shows a noticeably high sensitivity to the sample selection from the random variable provided to it. Therefore, the PADD method stabilizes its verdict with replication of the measurement, which is possible thanks to a reliable buffer of historical activations. The invariance of the model weights, in turn, preserves the repeatability of the transformations performed by the NN, which should lead to results of low-dimensional embeddings to be statistically dependent in the absence of changes in the conditional class probabilities of the stream – which can be associated with both real and virtual drift phenomenon. Consequently, the proposed method is not built around the observation of the decision boundary – as is the case with solutions basing detection on the evaluation of significant changes in the quality of processing – but presents the potential to register general changes in the distribution occurring regardless of a given label bias.

Figure 3.16 shows the intuition behind the method operation on the exemplary stream with 250 data chunks and a final NN layer with four outputs. The first line presents the image (probing of a model with a mesh grid covering two-dimensional feature space) of the NN output in the area sampled by data distribution. Red regions correspond

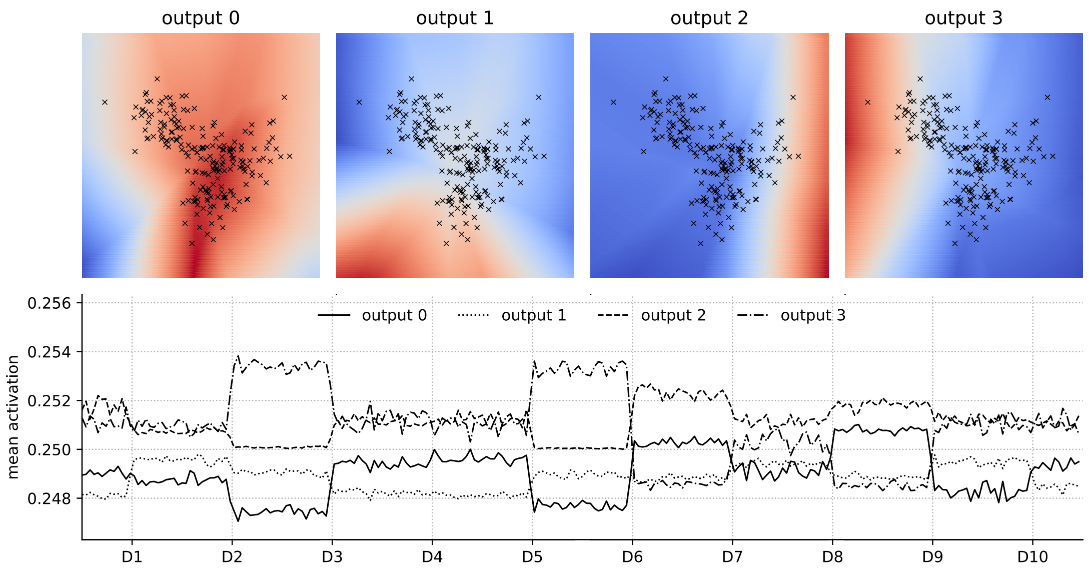

Figure 3.16: *An example of 2-dimensional data (black markers) presented in a context of ultimate layer activations of randomly initialized* NN *(top charts) and their mean activation (bottom chart) of four examined* NN *outputs during stream processing. Vibrant red areas in the top charts correspond to the high activation of a model, and vibrant blue to its low activation. The ticks on the horizontal axis of the bottom chart signal the moments of an abrupt concept change.*

to high activations, and blue to low activations. Random initialization of the weights causes diversified local landscapes in every dimension of the output space. In these areas, samples from one batch are marked with black markers. If drift occurs and the conditional distribution of a sampled data chunk changes, the structure of pseudo-supports in the recognized set maps this change within all or a part of the NN outputs. The second row of the figure presents the average activation values for batches in the data stream across all four outputs. The moments when drifts occur are clearly visible, additionally marked with chunk indices on the horizontal axis of the chart. In relation to the state of the network, some drifts are easier to identify than others – for example, the difference between the average output for the first drift (D1 on the horizontal axis) is less visible than for the next drift (D2).

3.3.1 The design of experiments

This subsection describes the experimental setup, the used data streams, and the goals of the specific experiments performed to thoroughly evaluate the proposed PADD approach. The evaluation first focused on analyzing the method's hyperparameters, to later compare the optimized method with *state-of-the-art* drift detectors.

Data streams The experimental evaluation was performed on synthetic data streams, where samples were characterized with various numbers of features, and the streams with a various number and types of drifts. The streams were processed in 250 batches, each comprising 200 samples. The method was designed for datasets with high dimensionality. Hence – the minimum value of 30 and the maximum of 90 features were selected for the experiments. Out of all features, 30% of them were informative. The streams were characterized by two types of concept drift dynamics – sudden and gradual changes, and from 3 to 15 concept drifts over the course of the stream.

Additionally, each stream with a specific configuration was generated ten times with varying random states of a generator. The complete configuration of data streams is presented in Table 3.13. For the first experiment a reduced pool of streams was used, where ten concept drifts were present. Three *drift detection error* measures were used to assess method operation in all experiments.

Table 3.13: *The configuration for generating synthetic data streams used for the experiments on PADD approach. The first column specified the data stream characteristics, and the second one the selected values.*

CHARACTERISTICS	CONFIGURATION
Number of chunks	250
Drift frequency	3, 5, 10, 15 drifts
Chunk size	200
Number of features	30, 60, 90 (30% informative)
Drift dynamics	*sudden*, *gradual*
Replications	10

Hyperparameter optimization The first experiment aimed to select appropriate hyperparameters of the proposed method. Out of the five available hyperparameters, only the two most critical were optimized: *alpha* and *threshold*. The remaining ones were fixed to values: number of network outputs $e = 12$, number of statistical test replications $t = 12$, and the sample size $s = 50$.

A neural network with a single hidden layer containing ten neurons and a *ReLU* activation function was used. For the *alpha* hyperparameter, 15 evenly sampled values from the range 0.03 to 0.2 were tested, and for the *threshold*, ten evenly sampled values from the range 0.1 to 0.3. The operation of the method with the indicated configurations was tested for the streams with ten drifts. The result of this experiment should indicate the range of values of these two hyperparameters for which the method effectively recognizes drifts. Since both examined hyperparameters indicate sensitivity to concept changes, their relationship should be visible.

Comparison with the reference methods The second experiment aimed to compare the proposed approach with reference methods. *State-of-the-art* supervised and unsupervised detectors were selected. If possible, the implementation of methods provided by the authors was used, or it was modified to allow processing streams in the form of data batches.

Table 3.14 presents all methods considered in the experiment. The first column shows the acronym of the method, the second presents the full name of the method and a reference to the article introducing this approach, and the third one – the category in the context of label access. The last column describes the hyperparameterization of the method used in the experiment.

Table 3.14: *The configuration of reference drift detectors for comparison experiment with* PADD *approach. The first two columns specify the method acronym and its name, the third one specifies supervised or unsupervised category, and the final column describes the selected hyperparameters.*

METHOD		CATEGORY	CONFIGURATION
MD3	*Margin Density Drift Detection* [202]	Unsupervised with label request	*threshold* set depending on number of features: 0.15 for 30 features, 0.1 for 60 features and 0.08 for 90 features
OCDD	*One-Class Drift Detector* [85]	Unsupervised	*percentage* hyperparameter set depending on problem dimensionality: 0.75 for streams with 30 features, 0.9 for 60 features and 0.999 in case of 90 features
ADWIN	*Adaptive Windowing* [24]	Supervised	default *delta* of 0.002, the base classifier used for error monitoring was Gaussian Naive Bayes
DDM	*Drift Detection Method* [70]	Supervised	default detection *threshold* of 3, the base classifier used for error monitoring was Gaussian Naive Bayes
EDDM	*Early Drift Detection Method* [12]	Supervised	default *beta* of 0.9, the base classifier used for error monitoring was Gaussian Naive Bayes

The default hyperparameters were selected for supervised methods, consistent with the implementation in the *scikit-multiflow* library [170]. For the remaining ones, the hyperparameters were manually selected according to the works introducing the methods or altered to effectively process the evaluated types of streams [85, 202].

3.3.2 Experimental evaluation

This subsection describes and analyzes the results of the experiments conducted to evaluate the proposed PADD method. The quality of drift detection was evaluated according to the *drift detection error* measures and the visual analysis of the detection moments.

Hyperparameter optimization The first experiment aimed to select the appropriate method hyperparameters. The analysis focused on two describing the method sensitivity – *alpha* value for statistical test significance and *threshold* value for method integration.

The results for three drift detection error measures and a single stream described by 30 features and characterized by sudden concept drifts are presented in Figure 3.17. The presented heatmaps indicate the $D1$, $D2$ and R errors, respectively. Saturated and dark cells describe low error values, while bright ones – a high error. Each heatmap's vertical axis describes the *alpha* (α) values, and the horizontal axis describes the *threshold* (θ).

The lower left area of each heatmap – for the low θ values and high α – describes a high sensitivity to changes present in the stream and is related to numerous redundant

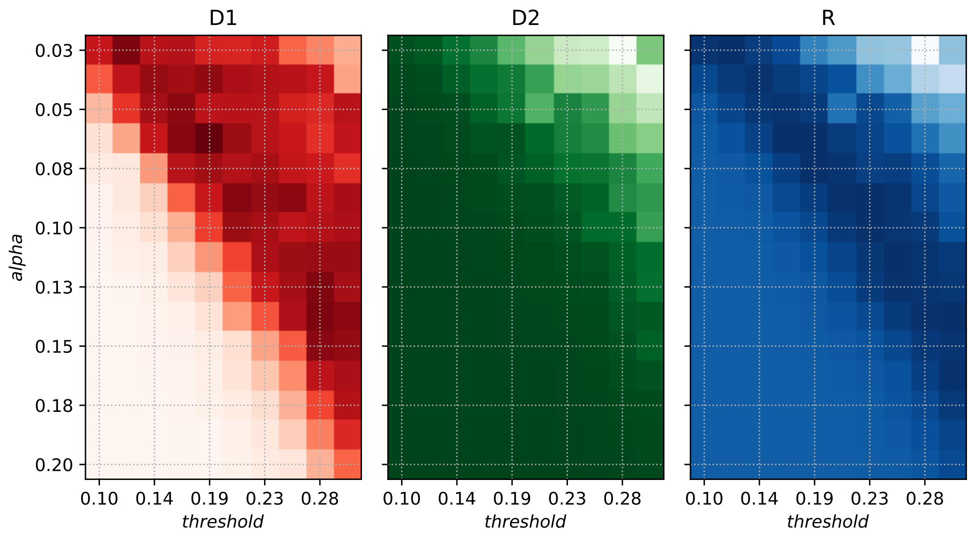

Figure 3.17: *Drift detection error measures for a single 30-dimensional data stream with sudden concept drifts, depending on the values of two critical hyperparameters – alpha and threshold, describing the method sensitivity.*

detections, visible in high $D1$ error and significant R error. The upper right area, in turn, describes the low sensitivity of a method – where all error values, including $D2$, are high. The best configuration of the method is lying between these two extremes.

Experiment results for all streams analyzed in the first experiment are presented in Figure 3.18. The RGB heatmaps show the combination of three *drift detection error* measures after their normalization. The results for streams with sudden drifts are presented in the first row, and for gradual concept change – in the second. The columns present various dimensionalities of the data – 30, 60, and 90 features, respectively.

After such a color combination of error values, the lowest errors in all three criteria is marked by dark colors. Similarly to the previously presented drift detection experiments, the measures for cases where the method did not provide any detections were calculated as for the detection in each data chunk.

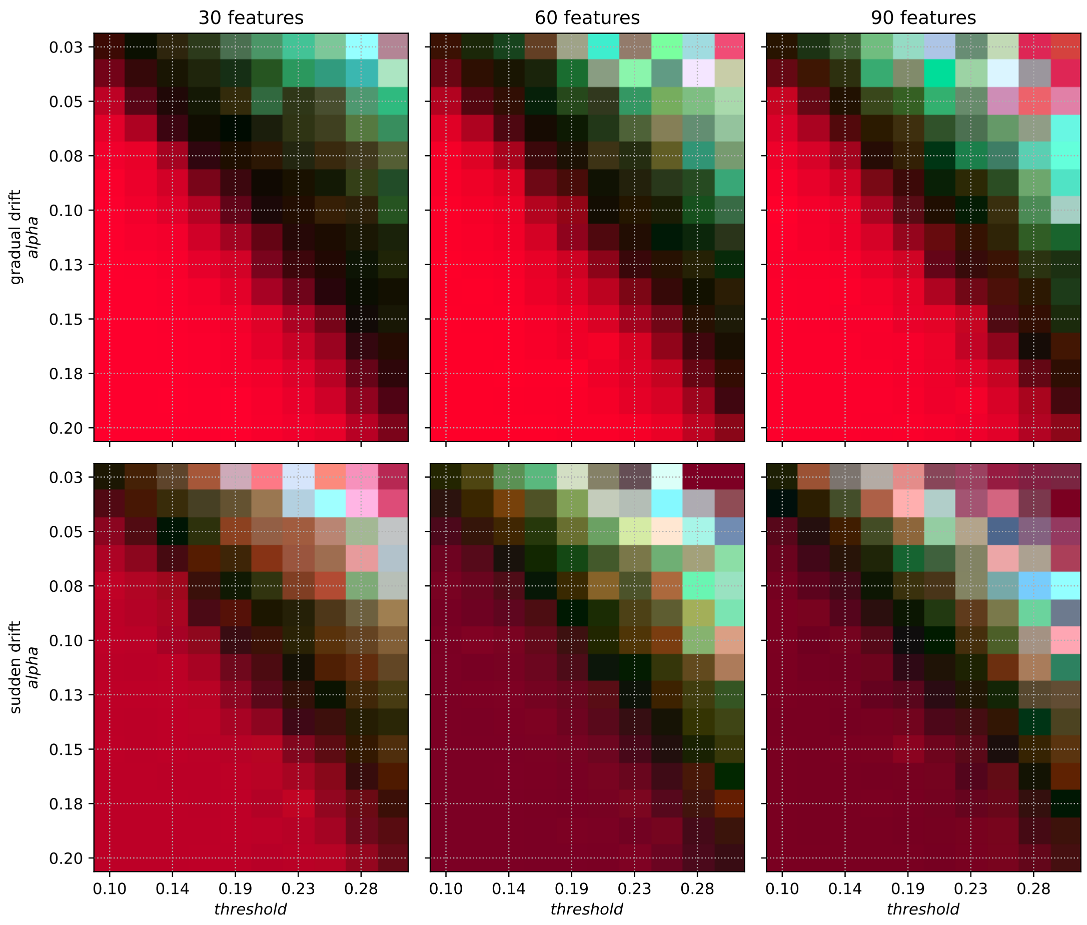

Figure 3.18: *The combination of drift detection error measures, normalized to a range 0-1 in each of three measures, presented as an* RGB *image. Black cells indicate the lowest error across all three measures, while bright ones – the highest errors.*

It is worth noting that the method's effectiveness is highly dependent on the selection of these two hyperparameters, and the lack of precise selection may result in excessive detection or failure to recognize drifts. Ultimately, the following hyperparameter combinations were selected for the comparison experiment: for *gradual* drifts $\alpha = 0.13$ and $\theta = 0.26$, while for *sudden* drifts $\alpha = 0.07$ and $\theta = 0.19$.

Comparison with the reference methods The second experiment compared the performance of the proposed approach with reference methods. The results for streams with sudden drifts are shown in Figure 3.19 and for gradual drifts in Figure 3.20.

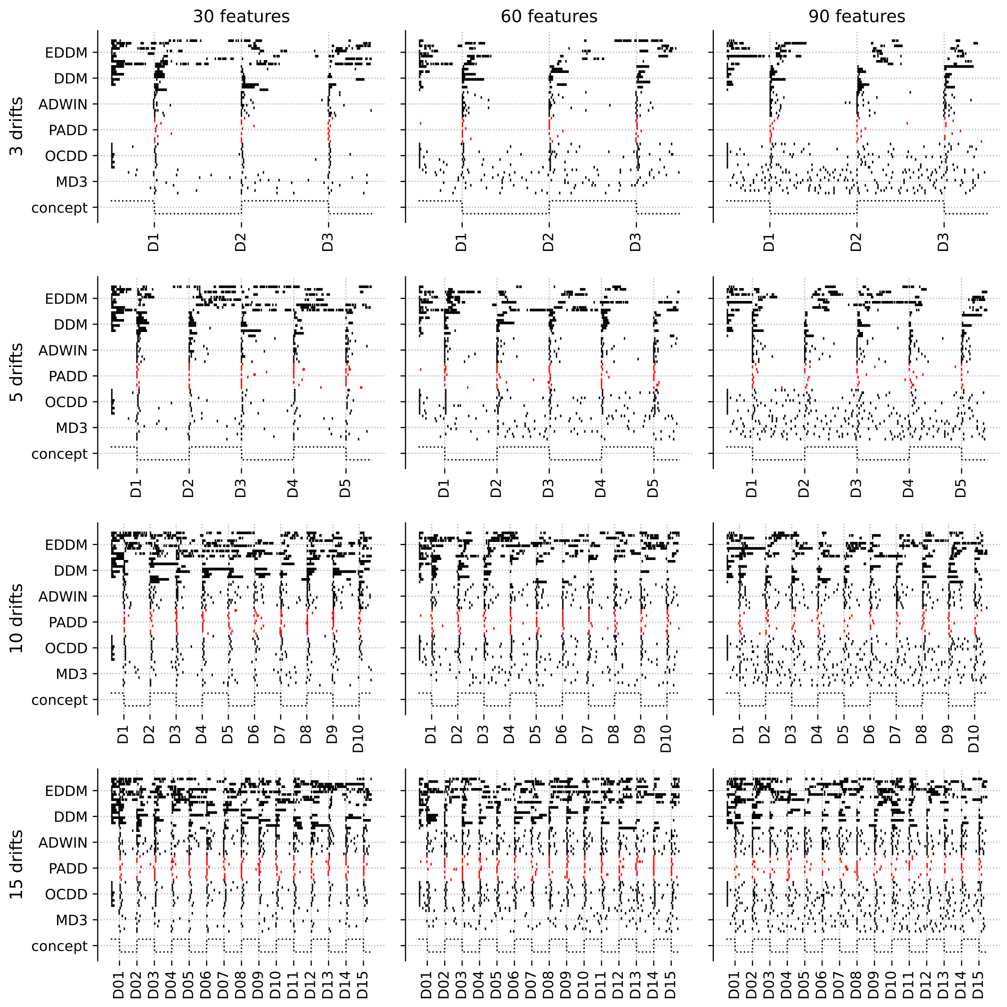

Figure 3.19: *The results of a second experiment comparing the* PADD *with reference approaches for all evaluated streams with* sudden *concept drift. The points indicate the detection moments of all evaluated methods across ten replications. The proposed approach is highlighted in red.*

The columns indicate the results for different numbers of features – from 30 to 90 – and the rows for different numbers of drifts present in the stream – from 3 drifts in the first row to 15 drifts in the last. On the horizontal axis of each plot, successive chunks of the data stream are visible, while the central moments of the actual drift are marked

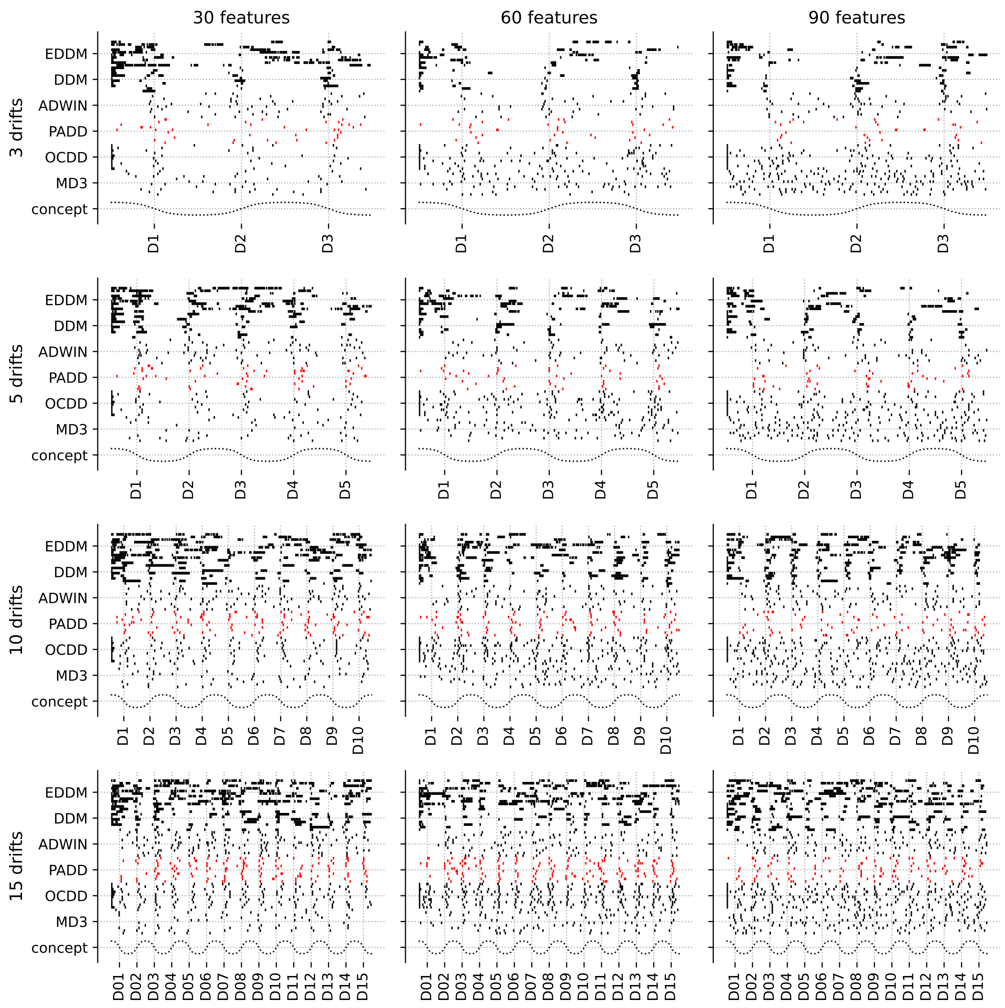

Figure 3.20: *The results of a second experiment comparing the* PADD *with reference approaches for all evaluated streams with* gradual *concept drift. The points indicate the detection moments of all evaluated methods across ten replications. The proposed approach is highlighted in red.*

with ticks and a grid. Each detection is marked with a single point. For emphasis, the proposed approach is shown in red. The consecutive lines show the results from subsequent replications for a given detector. The last row shows the concept drift dynamics.

The figures allow to notice frequent and redundant detections of DDM and EDDM methods, when drift is signaled in many consecutive data chunks, especially in the initial phase of a stream. This is consistent with the observations for the SDDE method, presented in Section 3.2. The last of the supervised detectors – ADWIN – also tends to repeatedly signal a single concept drift, but redundant detections occur sporadically, in single chunks. In the case of frequent changes, these redundant detections do not fade throughout the concept, which is typical for the high dynamics of changes. This may result from the emergence of a new concept before the classification quality within the current one has stabilized. The experiment did not reveal any changes in the quality of detection depending on the dimensionality of the problem in the case of supervised methods.

This may result from the mechanism of explicit drift detectors, not relying on the data distribution but on the classification quality of a baseline classifier – which must not have been significantly impacted by the problem dimensionality.

Among the reference unsupervised methods, the interesting behavior of the OCDD detector is visible, which signals a single redundant detection almost always in the initial processing phase. After some time, however, the detections reported by the method coincide with the drifts actually occurring in the stream. As the dimensionality of the problems increases, the method loses the ability to effectively recognize drifts, reporting many detections within stable concepts. The MD3 method also seems to lose the ability to recognize concept drifts effectively as the problem dimension increases, signaling more and more unnecessary, redundant alerts.

Table 3.15: *The results of comparison experiment expressing the D1 drift detection error measure. The columns present the evaluated methods, with the proposed* PADD *shown in red, and the rows – the type of data stream used in the experiment.*

			DETECTION FROM NEAREST DRIFT (D1)					
			MD3 (1)	*OCDD* (2)	*PADD* (3)	*ADWIN* (4)	*DDM* (5)	*EDDM* (6)
SUDDEN	3 DRIFTS	30F	14.272 6	14.458 6	**2.333** **1 2 5 6**	**3.506** **1 2 5 6**	15.865 —	21.002 —
		60F	18.718 —	15.645 1 6	**2.958** **1 2 5 6**	**5.292** **1 2 5 6**	13.607 —	22.879 —
		90F	20.025 —	17.159 6	5.467 1 2 6	**4.716** **1 2 5 6**	10.780 1 2 6	23.029 —
	5 DRIFTS	30F	5.814 6	7.907 6	2.842 2 5 6	**2.852** **1 2 5 6**	7.410 6	14.417 —
		60F	11.621 —	7.816 1 6	**2.034** **all**	3.665 1 2 5 6	6.999 1 6	12.983 —
		90F	11.628 —	9.676 1 6	**3.045** **1 2 5 6**	4.402 1 2 6	7.449 1 6	13.517 —
	10 DRIFTS	30F	3.781 5 6	3.458 5 6	**1.687** **all**	2.991 5 6	5.055 6	6.348 —
		60F	5.386 6	3.795 1 6	**2.010** **all**	3.693 1 6	4.363 6	6.448 —
		90F	5.590 6	5.320 6	**2.375** **all**	3.832 1 2 6	4.363 1 6	6.743 —
	15 DRIFTS	30F	2.094 5 6	2.669 5 6	**1.146** **all**	2.289 2 5 6	3.244 6	4.102 —
		60F	3.565 6	2.421 1 5 6	**1.464** **all**	2.651 1 6	3.022 6	4.283 —
		60F	3.525 6	3.673 6	**2.062** **all**	2.756 1 2 6	3.104 2 6	4.126 —
GRADUAL	3 DRIFTS	30F	15.676 2	21.578 —	14.650 2	**15.177** **2 6**	**13.068** **2 6**	21.293 —
		60F	18.768 —	16.815 —	14.748 —	15.411 —	15.498 —	19.758 —
		90F	21.223 —	17.427 1	16.322 —	**15.273** 1 6	**13.302** 1 6	21.285 —
	5 DRIFTS	30F	9.183 —	11.800 —	**7.564** **2**	**8.576** **2**	**7.052** **2**	9.938 —
		60F	11.656 —	**8.950** **1 6**	**7.860** **1 6**	**9.401** **1 6**	9.340 —	12.399 —
		90F	11.940 —	11.511 —	8.390 1 2	8.836 1 2	**6.233** **1 2 6**	11.839 —
	10 DRIFTS	30F	4.861 —	5.648 —	**4.242** **2 4 5 6**	5.304 —	5.462 —	5.728 —
		60F	6.636 —	**4.459** **1 4 6**	4.640 1 4	5.681 1	**4.487** **1 4 6**	5.731 1
		90F	5.699 4	6.284 —	5.888 —	6.420 —	**4.315** **all**	5.753 —
	15 DRIFTS	30F	**2.898** **2 4 6**	3.909 —	**2.854** **2 4 6**	4.102 —	3.223 2 4	3.813 —
		60F	4.067 —	**3.073** **1 4 5 6**	**2.955** **1 4 5 6**	4.599 —	4.181 —	3.915 4
		90F	**3.725** 4	4.336 —	**3.705** 4	4.542 —	**3.844** 4	4.070 —

Table 3.16: *The results of comparison experiment expressing the D2 drift detection error measure. The columns present the evaluated methods, with the proposed* PADD *shown in red, and the rows – the type of data stream used in the experiment.*

			DRIFT FROM NEAREST DETECTION (D2)					
			MD3 (1)	*OCDD* (2)	*PADD* (3)	*ADWIN* (4)	*DDM* (5)	*EDDM* (6)
SUDDEN	3 DRIFTS	30F	12.333 —	1.600 6	7.600 —	**0.433** **2 5 6**	9.033 —	9.700 —
SUDDEN	3 DRIFTS	60F	10.033 6	1.300 1 6	2.000 1 6	**0.700** **1 2 6**	19.533 —	27.700 —
SUDDEN	3 DRIFTS	90F	3.033 6	4.033 6	8.200 —	**0.533** **1 2 3 6**	3.733 6	15.133 —
SUDDEN	5 DRIFTS	30F	11.220 —	3.500 6	5.900 —	**0.680** **all**	6.720 —	11.920 —
SUDDEN	5 DRIFTS	60F	9.240 —	**1.100** **1 6**	**2.560** **1 6**	**0.900** **1 6**	12.360 —	18.000 —
SUDDEN	5 DRIFTS	90F	3.160 3 6	3.640 6	8.460 —	**0.220** **1 2 3 6**	16.620 —	13.860 —
SUDDEN	10 DRIFTS	30F	9.540 —	2.980 1	3.880 1	**1.290** **1 2 3**	12.290 —	6.140 —
SUDDEN	10 DRIFTS	60F	10.130 5	**1.530** **1 3 5 6**	4.190 1 5	**1.720** **1 3 5 6**	33.860 —	5.580 1 5
SUDDEN	10 DRIFTS	90F	4.620 —	3.370 3 6	7.040 —	**1.150** **1 2 3 6**	20.390 —	6.230 —
SUDDEN	15 DRIFTS	30F	13.580 —	3.060 1	2.900 1	**1.693** **1 5 6**	8.660 —	3.727 1
SUDDEN	15 DRIFTS	60F	10.840 —	**1.407** **1 3 5 6**	3.800 1 5	**1.720** **1 3 5 6**	17.760 —	3.733 1 5
SUDDEN	15 DRIFTS	90F	4.420 3 5	3.333 3 5	6.773 5	**1.107** **all**	18.260 —	3.747 3 5
GRADUAL	3 DRIFTS	30F	15.400 —	17.033 —	19.333 —	8.667 —	11.833 —	13.500 —
GRADUAL	3 DRIFTS	60F	9.800 4	**4.333** **1 3 4 6**	12.233 —	20.367 —	22.133 —	26.500 —
GRADUAL	3 DRIFTS	90F	**6.033** **3 6**	**5.400** **3 6**	22.600 —	13.267 —	12.300 —	13.133 —
GRADUAL	5 DRIFTS	30F	19.720 —	9.960 —	11.260 —	7.100 —	16.560 —	6.540 —
GRADUAL	5 DRIFTS	60F	13.840 —	**2.580** **all**	9.860 —	12.680 —	21.220 —	15.980 —
GRADUAL	5 DRIFTS	90F	**5.360** **3 4 6**	**5.760** **3 4 6**	14.780 —	15.140 —	15.580 —	20.840 —
GRADUAL	10 DRIFTS	30F	19.200 —	**5.670** **1**	**5.690** **1**	**7.170** **1**	20.780 —	**6.220** **1**
GRADUAL	10 DRIFTS	60F	11.880 —	**2.140** **all**	6.570 1	10.620 —	18.940 —	9.770 —
GRADUAL	10 DRIFTS	90F	**5.450** **3 4**	**4.980** **3 4**	12.250 —	9.050 —	10.820 —	7.580 —
GRADUAL	15 DRIFTS	30F	18.393 —	4.713 1 3	5.880 1	6.360 1	19.007 —	**3.640** **1 3 4**
GRADUAL	15 DRIFTS	60F	16.053 —	**1.940** **all**	4.913 1 4 5	8.960 —	20.660 —	6.520 1
GRADUAL	15 DRIFTS	90F	6.307 4	**4.427** **3 4**	7.667 —	8.380 —	19.020 —	**5.373** **3 4**

The proposed PADD method signals concept changes even in the case of frequent drifts, as can be seen in the last row for the case of 15 concept drifts. However, it can be observed that not all drifts are signaled, or their marking is delayed, in the case of high-dimensional streams visible in the last column. The results for streams with gradual drifts are similar. It can be seen that detections for all methods are more dispersed in the case of gradual drifts – this is either due to the detection in an early or later phase of drift or due to multiple signaling of the same change in the case of high sensitivity of the method, also noticed before and typical for the recognition of non-sudden concept drifts.

Tables 3.15 – 3.17 present the results of *drift detection error measures* for the conducted

Table 3.17: *The results of comparison experiment expressing the R drift detection error measure. The columns present the evaluated methods, with the proposed* PADD *shown in red, and the rows – the type of data stream used in the experiment.*

			RATIO OF DRIFTS TO DETECTIONS (R)					
			MD3 (1)	*OCDD* (2)	*PADD* (3)	*ADWIN* (4)	*DDM* (5)	*EDDM* (6)
SUDDEN	3 DRIFTS	30F	0.431	0.382	**0.100**	0.506	0.788	0.897
			5 6	5 6	**all**	5 6	6	—
		60F	0.602	0.636	**0.025**	0.521	0.947	0.838
			5 6	5 6	**all**	5 6	—	—
		90F	0.817	0.680	**0.050**	0.593	0.799	0.833
			—	1 5 6	**all**	1 2 5 6	—	—
	5 DRIFTS	30F	**0.269**	**0.295**	**0.190**	0.494	0.752	0.861
			4 5 6	**4 5 6**	**4 5 6**	5 6	6	—
		60F	0.376	0.517	**0.062**	0.505	0.652	0.797
			5 6	6	**all**	6	—	—
		90F	0.704	0.588	**0.158**	0.545	0.729	0.836
			6	1 6	**all**	1 2 6	—	—
	10 DRIFTS	30F	**0.259**	**0.183**	**0.116**	0.507	0.742	0.840
			4 5 6	**4 5 6**	**4 5 6**	5 6	—	—
		60F	**0.144**	0.390	**0.104**	0.492	0.494	0.813
			2 4 5 6	4 6	**2 4 5 6**	6	6	—
		90F	0.424	**0.401**	**0.285**	0.519	0.592	0.803
			5 6	**4 5 6**	**4 5 6**	6	6	—
	15 DRIFTS	30F	1.047	**0.116**	**0.084**	0.451	0.625	0.804
			—	**1 4 5 6**	**1 4 5 6**	1 5 6	6	—
		60F	0.453	0.344	**0.130**	0.409	0.502	0.771
			6	4 5 6	**all**	6	6	—
		90F	**0.205**	0.304	0.403	0.453	0.478	0.789
			all	4 5 6	6	6	6	—
GRADUAL	3 DRIFTS	30F	**0.329**	**0.350**	**0.355**	**0.456**	0.762	0.908
			5 6	**5 6**	**5 6**	**5 6**	6	—
		60F	0.576	0.656	**0.140**	0.332	0.744	0.788
			6	6	**all**	1 2 5 6	—	—
		90F	0.807	0.695	**0.240**	**0.427**	0.640	0.808
			—	1	**1 2 5 6**	**1 2 5 6**	1 6	—
	5 DRIFTS	30F	0.395	**0.241**	**0.215**	0.375	0.669	0.871
			6	**4 5 6**	**4 5 6**	5 6	6	—
		60F	0.468	0.590	**0.067**	0.227	0.627	0.791
			2 5 6	6	**all**	1 2 5 6	6	—
		90F	0.679	0.566	**0.237**	0.351	0.948	0.783
			6	1 6	**1 2 4 6**	1 2 6	—	—
	10 DRIFTS	30F	0.861	0.142	**0.185**	**0.227**	0.650	0.825
			—	**5 6**	**5 6**	5 6	6	—
		60F	**0.244**	0.457	**0.094**	**0.155**	0.582	0.708
			2 5 6	6	**2 5 6**	**2 5 6**	—	—
		90F	0.330	0.389	0.383	**0.098**	0.560	0.762
			5 6	5 6	6	**1 2 5 6**	6	—
	15 DRIFTS	30F	1.574	**0.083**	**0.165**	**0.101**	0.513	0.764
			—	**1 5 6**	**1 5 6**	**1 5 6**	6	—
		60F	0.610	0.371	**0.109**	0.319	0.574	0.702
			—	6	**1 2 4 6**	6	—	—
		90F	**0.188**	0.285	0.413	0.239	0.510	0.727
			3 5 6	5 6	6	5 6	6	—

comparative experiment. The heading of the proposed method is marked in red for emphasis. Table 3.15 presents the results of the $D1$ measure, which describes the average distance from each signaled detection to the closest actually occurring drift. Within this measure, the proposed approach achieves the best results for streams with sudden drift, being statistically dependent on the ADWIN detector for streams with three concept drifts. In the case of gradual drifts, it is difficult to clearly identify the method with the best performance – only EDDM is not among the statistically best methods in any stream type, which results from numerous redundant detections.

Table 3.16 shows the results for the $D2$ error measure, which describes the average distance from each actually occurring drift to the nearest detection. Here, within the sudden drifts, the dominance of the ADWIN detector is clearly visible, which achieved an error value below two for all cases, which means that drift was recognized on average after less than two chunks of the new concept. The proposed PADD method achieves higher average error values, which results from an individual, significantly delayed detections or lack of recognition of some drifts. Failure to recognize drift results in a significant increase in the average drift distance from detection. In the case of gradual drifts, the best results are achieved almost unambiguously by the OCDD method, which, as can be seen in Figure 3.20, is characterized by a high activity within the period of concept changes occurring in the stream.

Table 3.17 shows the results of the last measure of drift detection error – R, which describes the adjusted ratio of the number of drifts to the number of detections. Within this metric, the proposed PADD method achieves the best results for both sudden and gradual drifts. There are cases in which the MD3 and OCDD methods perform equally well in the case of sudden drifts. In the case of gradual drifts, the ADWIN method also presents low error values. The high quality of PADD expressed in the R error measure is due to the small number of redundant detections reported by the proposed approach compared to the reference methods.

It is again worth emphasizing that those three criteria should not be used independently to evaluate methods, and the juxtaposition of all three describes the proper and effective method operation.

The final results of the second experiment across all three drift detection error measures show that the proposed PADD *method achieves results that are competitive with the evaluated reference methods – both supervised ones, such as* DDM*,* EDDM *and* ADWIN*, and unsupervised ones, such as* MD3 *and* OCDD*. An important strength of the proposed method is the independence of the label access, which may be limited in the data streams with high-velocity.*

Chapter 4

Data stream classification

This chapter focuses on the task of *non-stationary data stream classification.* Since the *concept drift* remains an important and vivid topic in the area of data stream classification, the proposed methods utilize various *metafeatures* to improve the recognition quality of the non-stationary data by characterizing the processed concepts. The presented methods address the concept variability in the specific streaming conditions. Similarly to Chapter 3, the description of the three proposed methods is followed by the presentation of the experimental setup and the critical analysis of the obtained results.

The first of the presented approaches – *Metafeature Concept Selector* [120] – is dedicated to the data streams with recurring concepts. It utilizes the *statistical metafeatures* calculated on the disjoint data batches similarly to the scheme visible in the *Complexity-based Drift Detector*, presented in Section 3.1. The proposed method uses an ensemble with a pool of concept-specific classifiers.

The following method – *Prior Probability Assisted Classifier* [125] – was designed for the specific type of concept drifts, which affect the *prior probability distribution* of the data. The statistical metafeatures allowed for the prior probability estimation using the *Dynamic Statistical Concept Analysis* [124], which was integrated with a classifier to compensate the bias towards the majority class in the case of a significant temporal imbalance.

The final method introduced in this dissertation – *Certainty-based Architecture Selection Framework* [123] – addresses the classification task in the *computer vision data streams*, where the difficulty of the concept changes. The difficulty is being assessed using the output of a neural network, similar to the scheme followed by *Parallel Activations Drift Detector* described in Section 3.3. The proposed approach selects an appropriate neural network architecture based on the estimated difficulty of the data stream.

4.1 Metafeature Concept Selector

This section presents the *Metafeature Concept Selector* (MCS) method for non-stationary data stream classification. The proposed approach uses a pool of classifiers for the recognition task and a pool of one-class classifiers, denoted as metaclassifiers, for concept identification based on the data chunk metafeatures. The method was designed for data streams with recurring concepts, where re-identifying a concept known in the past could restore previous knowledge.

There are two primary components of the proposed method – the procedure itself and the factors used as the analyzed metafeatures. Both components are discussed below.

Procedure The method's operation is presented in the Algorithm 4. Three main hyperparameters are controlling the operation of the method:

- *threshold* (θ) – a threshold of meta-classifier decision function that needs to be reached to indicate a concept change;
- *minimal concept length* (λ) – a number of chunks since the last drift in which change of concept is not considered;
- *maximum training samples of one-class classifier* (Λ) – a number of meta-samples (chunk metafeatures) used to fit meta-classifier.

When the method is initialized, both the pool of classifiers and one-class meta-classifiers contain a single, unfitted model responsible for the recognition in the initial concept. The block dedicated to the concept change detection (lines 5:18) is skipped at the first chunk. However, the batch metafeatures are still calculated (line 3). These values are used to train the single meta-classifier (Ψ^o) in the pool. Training of the classifier dedicated to the recognition task (Ψ) is also performed.

At subsequent chunks, if the lower limit of the chunks (λ) in the concept has been reached, drift detection is additionally performed (line 7). First, after calculating the metafeatures, a check is performed to see if the meta-classifier Ψ^o corresponding to the current concept j recognizes the batch metafeatures as instances of a known class. The detection of the new concept is determined by the decision function value of the meta-class classifier. The method then verifies whether the batch is from a new concept that has not been recognized previously or whether the concept has already occurred in the past. A new concept is recognized if the batch support of no meta-classifier exceeds the *threshold* value (line 10). Otherwise, the meta-classifier with the maximum decision function is recognized as dedicated to the current concept (line 14).

Algorithm 4 Pseudocode of the *Metafeature Concept Selector*

```
Input:
    $\mathcal{DS} = \{\mathcal{DS}_1, \mathcal{DS}_2, \ldots, \mathcal{DS}_k\}$ – data stream
                                                            ▷ Hyperparameters
    $m$ – set of metafeatures
    $\theta$ – threshold
    $\lambda$ – minimal concept length
    $\Lambda$ – maximum training samples of one-class classifier
                                                ▷ Base classifiers and ensembles
    $\Psi$ – base classifier
    $\Psi^o$ – base one-class classifier
    $\Pi$ – ensemble of classifiers
    $\Pi^o$ – ensemble of one-class classifiers
                                                            ▷ Parameters
    $j$ – current concept identifier
    $k_\delta$ – index of the most recent concept change
    $\mathcal{MF}$ – metafeature values for successive chunks

 1: for all $\mathcal{DS}_k \in \mathcal{DS}$ do          ▷ Calculate metafeatures for current data batch
 2:     for all $m_n \in m$ do
 3:         $\mathcal{MF}_n \leftarrow m_n(\mathcal{DS}_k)$
 4:     end for
 5:     if $k \neq 1$ then          ▷ Calculate decision function values of one-class classifiers
 6:         $s \leftarrow \Pi^o(\mathcal{MF})$                        ▷ Concept recognition
 7:         if $s_j < -\theta$ and $k - k_\delta > \lambda$ then
 8:             signal concept drift
 9:             $k_\delta \leftarrow k$                          ▷ Recognition of a new concept
10:             if $max(s) < -\theta$ then
11:                 $\Pi^o \leftarrow \Pi^o \cup \{\Psi^o\}$
12:                 $\Pi \leftarrow \Pi \cup \{\Psi\}$
13:                 $j \leftarrow |\Pi|$
14:             else                                  ▷ Recognition of a past concept
15:                 $j \leftarrow argmax(s)$
16:             end if
17:         end if
18:     end if                                                ▷ Classifier fitting
19:     $\Psi_j \leftarrow \Psi_j(\mathcal{DS}_k)$
20:     $\Psi^o_j \leftarrow \Psi^o_j(\Lambda$ random values from $\mathcal{MF}$ since $k_\delta)$
21: end for
```

As the final processing step, the classifiers corresponding to the current concepts in the Ψ and Ψ^o pools are fitted with the current data chunk and its metafeatures, respectively (lines 19:20).

The method is designed to detect new concepts based on the chunk metafeatures, and to restore the knowledge accumulated in the past in the case of recurring concepts. If the concept recurs, the particular classifier is also incrementally updated with the new, labeled data samples.

Factors Table 4.1 presents the default set of metafeatures, identified as promising in the task of concept recognition and used in the presented research. The method was designed for the statistical category of metafeatures, as the preliminary research showed their potential in concept identification task [121]. Moreover, the selected set of metafeatures allows for fast computation, making it suitable for data stream processing. It is worth mentioning that a presented method can be adapted to any set

Table 4.1: *A set of statistical metafeatures considered as default and used for the experiments on the* MCS *method. The first column presents the name of a metafeature, while the second one describes its semantics.*

METAFEATURE	DESCRIPTION
mean	Mean of each feature's values
median	Median of each feature's values
truncated mean	Truncated mean (discards outlier observations) of values among features
gravity	Distance between minority and majority classes center of mass – the average value of each feature between instances of the same class [8]
covariance	Absolute value of the covariance of feature pairs
correlation	Absolute value of the correlation of feature pairs
canonical correlations	Canonical correlations calculated between the features and the labels
wilks lambda	Wilks' Lambda value, calculated based on the eigenvalues related to the canonical correlations between the features and the labels [149]
pillai's trace	Pillai's trace – sum of the squared canonical correlations [184]
lawley-hotelling trace	Lawley-Hoteling trace, calculated from the eigenvalues related to each canonical correlation [141]
roy's largest root	Roy's largest root, formulated using the largest eigenvalue associated with the canonical correlations [196]

of measures, allowing it to be used for unsupervised detection if the specified measures do not require labels.

All metafeatures were integrated by the *mean* summarization function [192], which resulted in exactly one value describing the data batch for each used metafeature. Considering all measures described in the table, the method calculated and analyzed the total

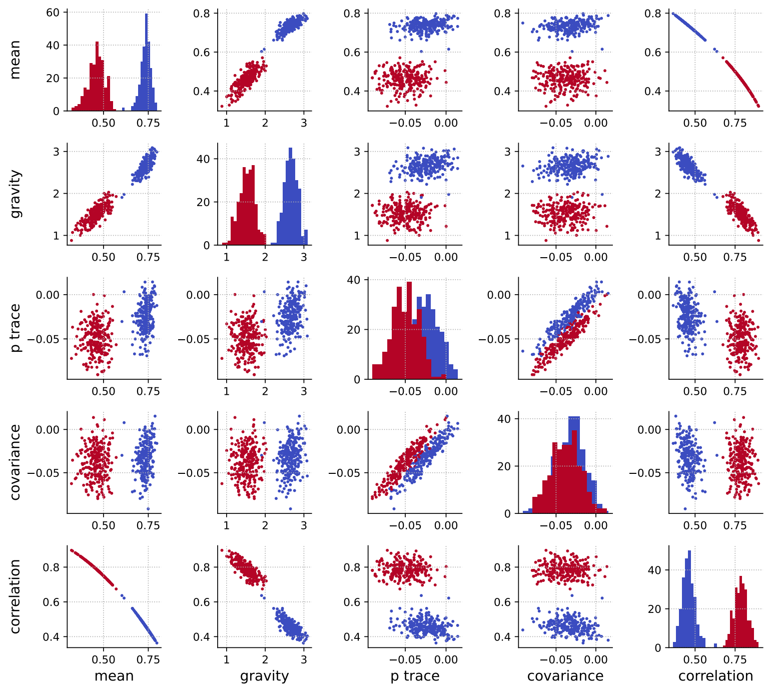

Figure 4.1: *The values of selected metafeatures calculated for consecutive data chunks with color identification of concept membership returned by the proposed approach. Each point indicates metafeatures for a single batch, and their colors identify the concept from which the sample originates.*

of 11 metafeatures for each data batch. The selected measures do not require the initialization and utilization of underlying recognition methods (which was necessary, for example, for problem complexity measures [154]), which makes them suitable for the processing of data streams, where the low computational complexity is of critical value.

Figure 4.1 shows how the selected metafeatures allow concept description. Five out of the total number of 11 metafeatures presented in Table 4.1 were selected to allow for clear presentation. Each point describes the metafeature values for a single chunk of the data stream.

For this preliminary visualization, a 20-dimensional data stream with seven recurring *sudden* concept drifts was used. The colors indicate the two recurring concepts recognized by the proposed method. The points presented in the Figure show how the metafeatures describing the two recurring concepts cluster into two groups, allowing for the detection of a concept change and its identification.

4.1.1 The design of experiments

This subsection describes the used data streams and the goals of specific experiments designed to analyze the operation of the proposed MCS method and to compare the approach with baselines. Since the proposed method solves the specific metaproblem of concept identification, the evaluation partially relies on the use of clustering metrics to support the employment of traditional accuracy-based evaluation.

Data streams Due to the low availability of data streams with recurring concepts and the general advantages of performing evaluations on synthetic data, experiments were performed on fully synthetic data streams generated with *stream-learn* library [133].

Table 4.2: *The configuration of a generation function for the data streams analyzed in* MCS *experiments. The first column specifies the characteristics of a generator, and the second one specifies the selected values.*

CHARACTERISTICS	CONFIGURATION
Number of chunks	500
Chunk size	250
Drift frequency	5, 7, 9, 11 drifts
Drift recurrence	True
Number of features	10, 20, 30 (30% informative)
Drift dynamics	sudden
Replications	10

Table 4.2 presents the details of the data generator configuration. Each stream consisted of 500 data chunks, and each data chunk aggregated 250 samples. The samples were described by various number of features – from 10 to 30, of which 30% were informative. The data streams were characterized by various numbers of drifts – from 5 to 11 drifts of sudden type. The recurring drifts were considered to thoroughly evaluate the method's ability of concept identification.

Finally, ten replications of each stream with described characteristics were generated, allowing for a reliable analysis of the results.

In the first part of the study, which focused on selecting the method's hyperparameters, a reduced number of streams were analyzed, considering only streams with seven concept changes.

Considering only sudden drifts was justified by the clear strategy of evaluating the method, in particular the moments of detection of concept changes and the matching of the recognized distribution (concept) to the one present in the past. As already noted during the evaluation of concept drift detectors, the methods tend to recognize the transition period between concepts as an independent one. Depending on the sensitivity of a given drift detection method, the onset of changes was noticed in an earlier or later phase of changes – or even multiple stages of gradual and incremental changes were recognized.

In the case of the proposed method, it is not the detection itself that is taken into account during the evaluation but the identification of the concept. In order to compare the concepts identified by the method with those actually occurring in the stream, it is worth eliminating the possibility of recognizing a transitional concept. Such behavior of the method would not indicate its incorrect operation but could make the results of the experiments inconclusive.

Hyperparameter optimization The first experiment aimed to optimize the most significant hyperparameter of the proposed method (θ), indicating the method's sensitivity to changes occurring in the monitored metafeatures. The remaining hyperparameters remained constant at *maximum training samples of one-class classifier* $\Lambda = 25$ and *minimal concept length* $\lambda = 5$.

The experiment focused on selecting the θ hyperparameter, since the other did not appear to be critical for the method's operation in the preliminary research. The evaluation concerned a hundred values uniformly sampled in the range from 0.5 to 4. The *one-class* SVM algorithm with RBF kernel was used as the default metaclassifier in the proposed method. In the first experiment, the Gaussian Naive Bayes (GNB) classifier was used as the default recognition method.

In the first experiment, two metrics were analyzed: *Rand index* (RI) [102] of the concept identification task and *accuracy* of the classification task. The first metric is originally dedicated to the assessment in the clustering task. Its selection was motivated by the possibility of comparing concept identifiers recognized by the method with those actually occurring in the stream. The metric is described by Equation (4.1), where a is the number of pairs of objects belonging to the same cluster as in ground truth, and b is the number of such pairs belonging to different clusters than specified by ground truth.

$$RI = \frac{a+b}{\binom{n}{2}} \tag{4.1}$$

The binomial coefficient in the fraction's denominator indicates the number of pairs analyzed during the evaluation. The value of this metric in case of an ideal matching of labels and cluster ground truth results in a value of 1. In case of a complete discrepancy between indicated clusters and a ground truth – the *Rand index* value equals 0.

In this case, the *Rand index* compares the recurring concept identifiers determined by the method to those calculated for the synthetic stream as a *ground truth*. This strategy of evaluation has been illustrated in Figure 4.2, where the indexes 0 and 1 on the vertical axis indicate concepts present in the stream, and the plots – the actual concept present in the stream (black dotted line) or the concept identified by the method (red solid line). The comparison of actual and recognized concept identifiers using *Rand index* allows for the evaluation of the method's ability of concept identification, limiting bias related to the selection of a base classifier.

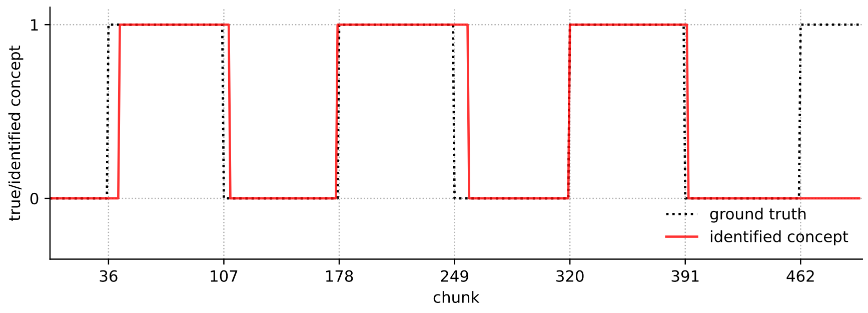

Figure 4.2: *The figure presenting the motivation of using clustering metrics for concept identification assessment. The horizontal axis shows the chunks of the data stream, with the concept drifts marked with ticks. The vertical axis indicates the recurring concept described by the data in specific chunks. The plots present the actual concept (black dotted line) and the identified one (red solid line).*

The classification quality measure should confirm the assumption that effective concept recognition brings high recognition quality. Intuitively, for a method to correctly recognize concepts, each classifier should be specialized to the task considered in each of the following concepts. However, in case of incorrect recognition, e.g., resulting from too late recognition of a change, the classifiers are trained using patterns originating from the concept for which they were not adapted – resulting in a decrease in the classification quality. Recognizing a concept change but failing to correctly identify it as seen in the past results in a new, redundant classifier being added to the pool. This behavior should also decrease the recognition quality compared to a perfect recognition of the past,

Table 4.3: *Table presenting the baseline approaches taken into account in the comparative experiment of* MCS *method. Each listed method was evaluated alone and compared with the version embedded into* MCS. *The first columns present the method acronym and its name, and the final column describes its configuration.*

METHOD		CONFIGURATION
GNB	*Gaussian Naive Bayes*	the default variance smoothing of 1e-9
MLP	*Multilayer Perceptron*	a single hidden layer consisting of 10 neurons, the default *ReLU* activation function, *Adam* solver; the incremental learning scenario applied a single iteration (epoch) per data batch
HTC	*Hoeffding Tree Classifier* [103]	the default Information Gain split criterion, the default number of instances a leaf should observe between split attempts equal to 200

recurring concept, as an adaptation of the classifier is required instead of applying previous knowledge.

Comparison with reference methods The second experiment compared the method performance with the baseline algorithms that allow incremental training. All reference methods considered in the evaluation are described in Table 4.3.

All those reference approaches were evaluated as a single classifier, as well as used as a base classifier embedded into the proposed MCS approach. In the second experiment, only the classification accuracy of all methods was collected and compared.

In case of a correct concept identification, the benefits of using MCS should be especially visible for MLP and HTC, as those methods are dedicated to incremental learning and require multiple training iterations until convergence – hence, the adaptation to consecutive concepts without a dedicated mechanism could potentially hinder the training process.

4.1.2 Experimental evaluation

This section presents the outcomes of the experiments and analyzes their results. The experiments aimed to analyze the sensitivity of MCS method to the concept change and concept re-identification and to compare the proposed method with baselines. The second experiment allows for the general evaluation of the proposed mechanism, regardless of the method-specific strategies employed in the wide range of sophisticated ensemble classifiers.

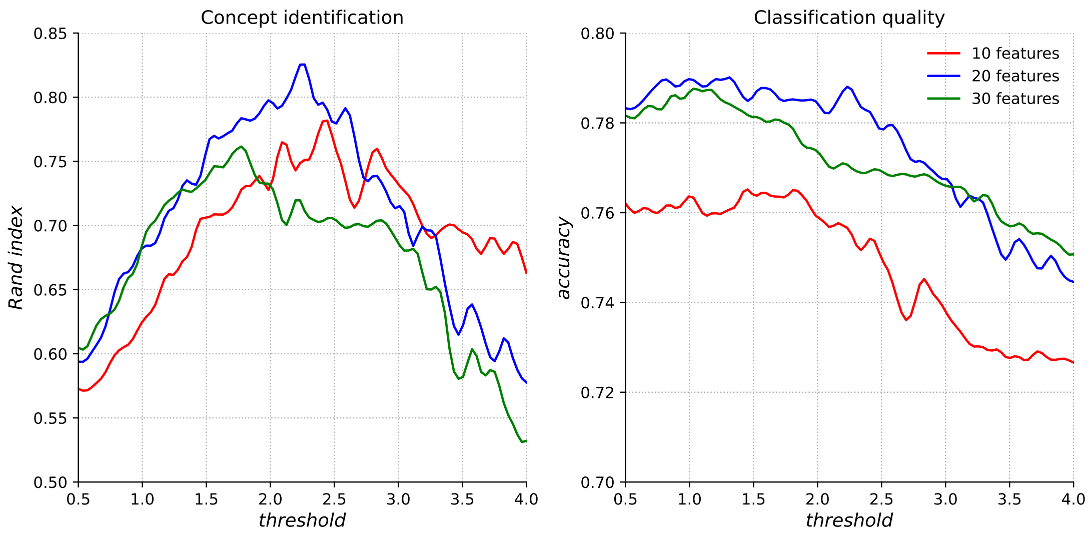

Figure 4.3: *Results of the threshold hyperparameter selection experiment. The average* Rand index *of the concept identification task is presented on the left side, while the average classification accuracy is on the right. The results were filtered with a Gaussian filter.*

Hyperparameter optimization The first experiment aimed to select the appropriate *threshold* (θ) hyperparameter of the method. The results presenting the average metrics for each threshold value are presented in Figure 4.3.

The three plots present results for three different dimensionalities of the data – from 10 to 30 features. The left side of the figure presents the results of a concept identification task, measures with *Rand index*, where the objective was to correctly identify the concept based on the calculated statistical metafeatures. The right side of the figure presents the classification accuracy results, used to evaluate how the correct concept identification impacts the average recognition quality. The horizontal axis of each subplot shows the values of the threshold hyperparameter used in the method configuration.

The results presenting the quality of concept identification show that for each dimensionality of the stream, there is an area with an increased *Rand index*. As the number of features increases, the suboptimal value of the hyperparameter slightly decreases – from about $\theta = 2.2$ for streams with ten features to about $\theta = 1.6$ for streams with 30 features.

The best classification quality is usually achieved for threshold values lower than those for which the *Rand index* is the highest. This is because the GNB classifier adapts to a new concept almost immediately, and the frequent addition of a new classifier (typical of a highly reactive method at a low threshold) results in a high average classification quality. However, this behavior of the method affects the size of the pool of stored classifiers, causing memory overhead. Moreover, when using other classifiers, such as MLP,

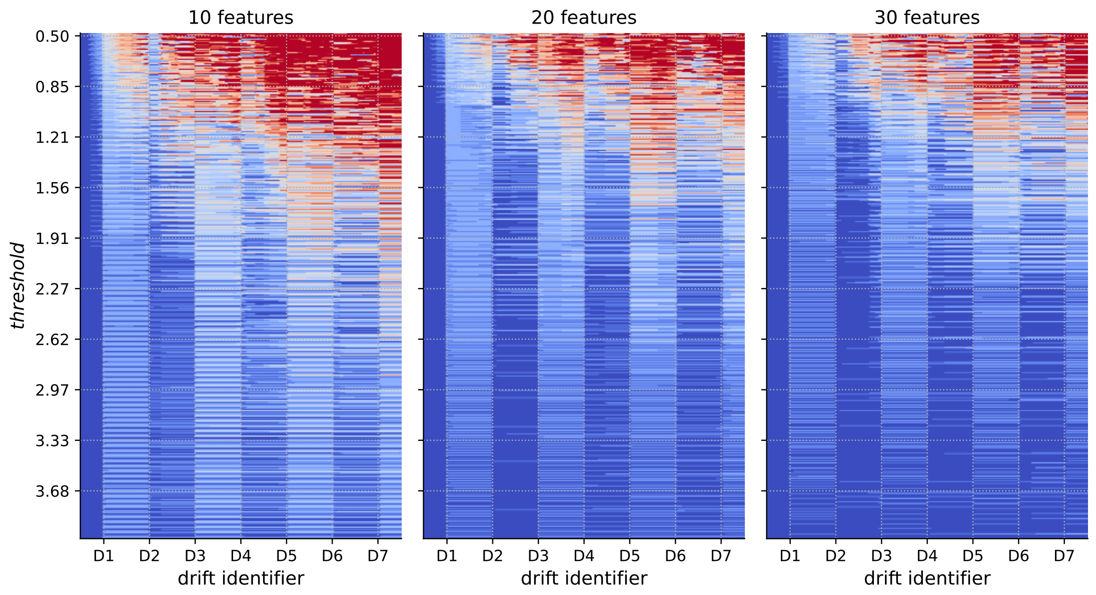

Figure 4.4: *Results of the threshold hyperparameter selection experiment – the visual representation of identified concepts. The columns present the results for different data stream dimensionalities – from 10 to 30 features. The color describes the index of the recognized concept – with the deep blue describing the index of* 0 *and saturated red describing the concepts with high index.*

which requires many iterations to converge, frequently adding a classifier to the pool results in a decrease in average classification quality – as none of the newly added classifiers are fully adapted to the problem being processed. For the above reasons, the main criterion considered when selecting the *threshold* hyperparameter was the *Rand index* describing the correctness of concept identification.

Additionally, Figure 4.4 visually presents the concept identification results for different dimensions of data streams. The horizontal axis of each subplot describes the moments of concept drift, while the vertical axis describes the values of the threshold hyperparameter. The stream replications were stacked in rows so that for each threshold value, ten repetitions were presented. In the chart, the color of an area is connected with the identification of the selected concept. In the case of the streams studied, only two concepts occurred interchangeably. A method that works correctly should, therefore, identify only two concepts described in shades of blue, in which the moments of transition coincide with the moment of drift occurrence visible on the horizontal axis.

The figure shows that low values of the *threshold* result in a high frequency of signaling concept changes and the inability of the methods to correctly recognize their recurrence – hence the red area at the top of each heatmap. High hyperparameter values result in an inability to recognize changes or recognition of subsequent concepts only in some stream replications – which is especially visible for streams with 30 features.

For further experiments, the following default values of the hyperparameter were selected: $\theta = 2.2$ for streams with ten features, $\theta = 2.0$ for streams with twenty features, and $\theta = 1.6$ for the highest dimensionality tested. The other hyperparameters remained unchanged.

Comparison with reference methods The second experiment compared the recognition quality using the baseline methods with the recognition quality using the proposed classifier selection approach.

Figure 4.5 shows the results of the comparison experiment. The classification qualities were averaged over ten stream generation iterations and smoothed using a Gaussian filter. The columns show streams with three different dimensionality – from 10 features in the left to 30 features in the right column. The rows show the results for different numbers of drifts – from 5 drifts in the first row to 11 in the last one. The methods are presented using different colors. For each tested baseline method, the classification quality using

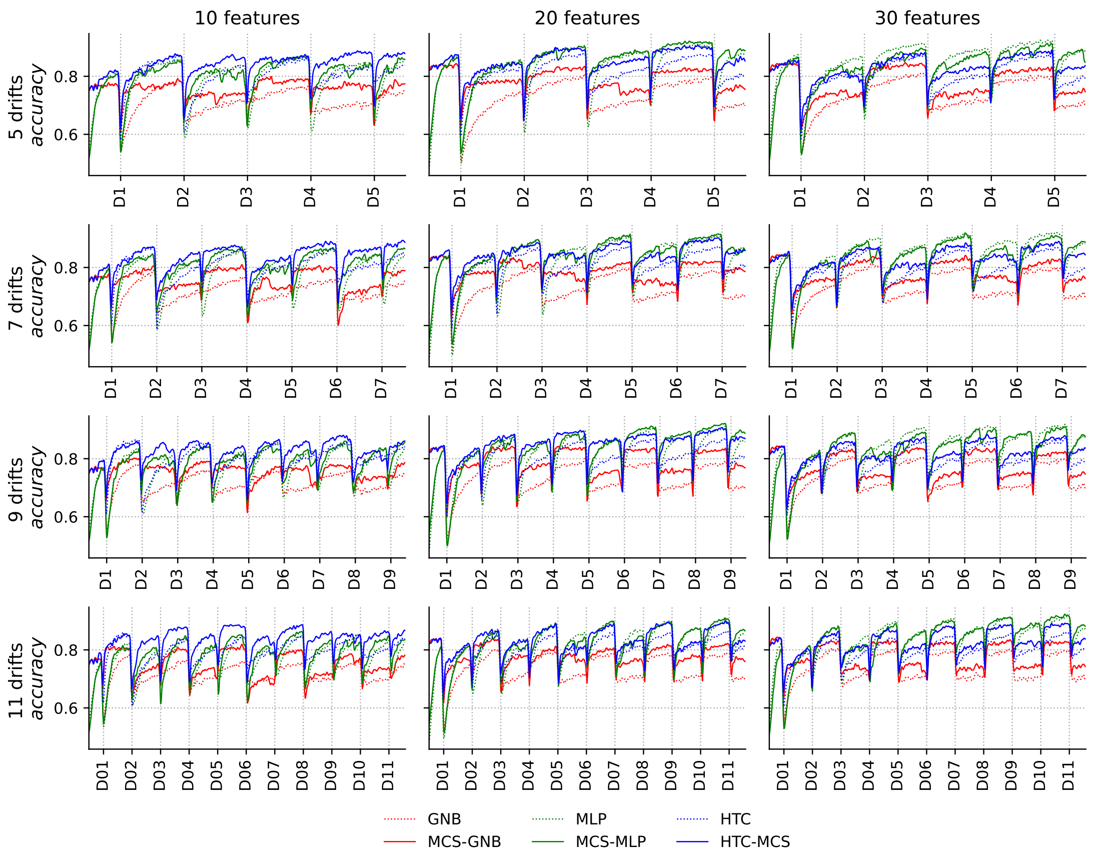

Figure 4.5: *The results of a comparative experiment. The accuracies of baseline methods are shown with dotted lines, and methods embedded into the proposed* MCS *approach with solid lines. The columns present different data dimensionalities, and the rows – the different number of drifts occurring in the stream.*

the baseline alone is shown using a dotted line, and for the baseline integrated with the proposed MCS approach, with a solid line.

When concept drift occurs, a clear drop in recognition quality is visible for all methods. It is also easy to notice the degeneration of recognition quality using the GNB algorithm in the case of recurring concept drifts. This method, not equipped with a forgetting mechanism, which is natural for solutions such as MLP, is not able to fully adapt to the current concept even with multiple training iterations. For this reason, the proposed approach brings the best benefits compared to GNB as a base method. However, the runs marked with a solid red line show individual moments of a drop in classification quality, resulting from individual, incorrect recognitions of a concept change within the replication or its incorrect identification.

Both MLP and HTC as base methods achieve good classification quality on their own, however, returning to a high recognition quality after drift occurs takes up to several dozen chunks of data using this method. The results show that the proposed MCS approach integrated with these methods can allow for faster switching of the classifier

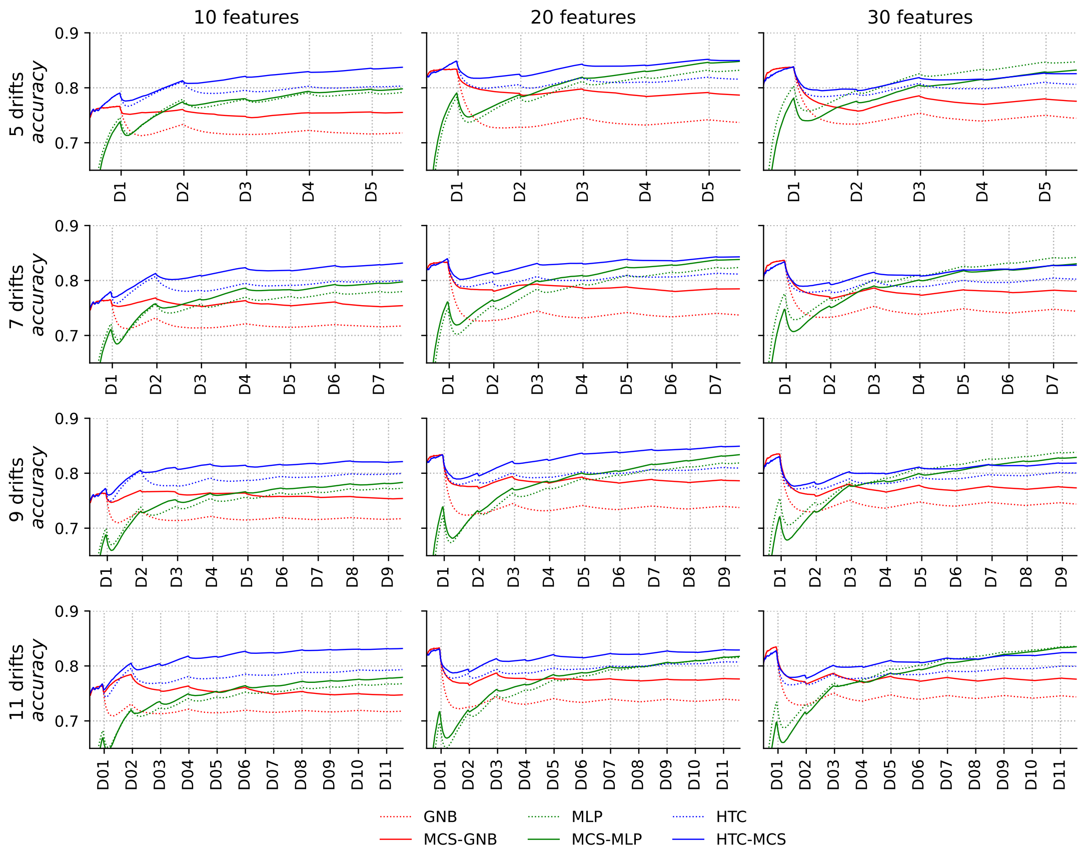

Figure 4.6: *The results of a comparative experiment with the accumulated classification accuracy. The results of baseline methods are shown with dotted lines and for methods embedded into the proposed* MCS *approach with solid lines.*

to one previously adapted to the current concept. Hence, the most significant benefit of the proposed approach is usually observed immediately after the concept drift occurs. There are occasional periods during processing for which the baseline MLP achieves better quality than the MCS-integrated MLP. Similar to the case of GNB, this is due to incorrect identification of the occurring concept.

Additionally, the accumulated results are presented in Figure 4.6 to facilitate the interpretation of the average recognition quality. The value on the plot in a given chunk corresponds to the average value of the average results obtained up to that point in the stream.

The figure shows that, ultimately, after the entire stream processing period, the average quality achieved by the MCS-integrated method is better than the base method for the GNB and HTC algorithms. For the MLP algorithm, this is visible for data with 10 and 20 dimensions. In the case of 30 dimensions, after averaging, there are cases when MLP alone achieves better average results than the MCS-integrated method. However, with very frequent drifts (11 drifts over 500 chunks in the presented case), the MCS-MLP method is equal in quality to the base MLP method, which may result from quality drops immediately after concept changes and predict that with an even greater frequency of changes, the proposed approach brings greater benefits.

The averaged results are also presented in Table 4.4. The table contains the results of the Student's T-test with a significance level of $alpha = 5\%$. Below the average result for each method and each stream type, the identifiers of the methods from which the data is statistically significantly better are presented. The columns presenting the methods integrated with the proposed approach are highlighted in red. Additionally, if there was a statistically significant difference between the pair (baseline compared to MCS), the x marker is shown for the leading approach.

As expected, the GNB algorithm, which is not adapted to incremental learning in data streams, performs the worst in the classification task. Integrating the base GNB with the MCS method brings statistically significantly better results in each stream type, even dependent on MLP in the case of low-dimensional streams. The baseline of MLP method achieves statistically significantly the best results, or the results are statistically dependent on MCS-MLP in the case of streams with the highest dimensionality. Since the largest drops of recognition quality were visible shortly after the concept drifts, combined with the relatively long time of MLP convergence to a new concept, the use of MCS brings the largest benefits for the streams with frequent concept drifts. The HTC method integrated with MCS (MCS-HTC) brings the best recognition quality in 10- and 20-dimensional streams, almost always statistically significantly better than the baseline HTC algorithm.

Table 4.4: *A table presenting the results of comparison experiment for the* MCS *approach. The columns indicate the methods – with the ones integrated with* MCS *highlighted in red – and the rows the types of data streams.*

		GNB (0)	*MCS-GNB* (1)	*MLP* (2)	*MCS-MLP* (3)	*HTC* (4)	*MCS-HTC* (5)
10 FEATURES	D: 5	0.719	0.757	0.793	0.800	0.805	**0.839**
		—	0	0	0 1	0 1 2	**all**
		—	x	—	—	—	x
	D: 7	0.719	0.756	0.781	0.799	0.802	**0.833**
		—	0	0	0 1 2	0 1 2	**all**
		—	x	—	x	—	x
	D: 9	0.719	0.756	0.774	0.785	0.801	**0.823**
		—	0	0	0	0 1 2	**0 1 2 3**
		—	x	—	—	—	—
	D: 11	0.719	0.749	0.769	0.781	0.795	**0.833**
		—	0	0	0	0 1 2	**all**
		—	—	—	—	—	x
20 FEATURES	D: 5	0.738	0.788	0.834	**0.850**	0.817	**0.851**
		—	0	0 1 4	**0 1 2 4**	0 1	**0 1 2 4**
		—	x	—	x	—	x
	D: 7	0.739	0.786	0.825	**0.840**	0.813	**0.845**
		—	0	0 1	**0 1 4**	0 1	**0 1 4**
		—	x	—	—	—	x
	D: 9	0.739	0.788	0.820	0.835	0.811	**0.851**
		—	0	0 1	0 1 4	0 1	**all**
		—	x	—	—	—	x
	D: 11	0.739	0.778	**0.817**	**0.819**	**0.809**	**0.831**
		—	0	**0 1**	**0 1**	**0 1**	**0 1**
		—	x	—	—	—	—
30 FEATURES	D: 5	0.746	0.777	**0.849**	0.834	0.808	0.827
		—	0	**all**	0 1 4	0 1	0 1 4
		—	x	x	—	—	x
	D: 7	0.746	0.782	**0.844**	**0.832**	0.804	**0.830**
		—	0	**0 1 4**	**0 1 4**	0	**0 1 4**
		—	x	—	—	—	x
	D: 9	0.745	0.775	**0.840**	0.830	0.802	0.820
		—	0	**0 1 4 5**	0 1 4	0 1	0 1 4
		—	x	—	—	—	x
	D: 11	0.745	0.778	0.837	**0.837**	0.801	0.826
		—	0	0 1 4	**0 1 4 5**	0 1	0 1 4
		—	x	—	—	—	x

Considering the obtained results, the proposed MCS *approach of selecting classifiers based on statistical metafeatures of data batches brings benefits in the case of processing data streams compared to using the baseline classifiers. The degree of such an improvement depends on the mechanism embedded into the baseline learner – where the methods without the direct adaptation for non-stationary data benefit the most (like* GNB*), as well as the data stream dimensionality and the frequency of concept changes.*

4.2 Prior Probability Assisted Classifier

This section describes the *Prior Probability Assisted Classifier* (2PAC) method that employs the estimated prior probability information to improve the classification quality in the statically and dynamically imbalanced data stream classification task. The prior probability estimation utilized by the proposed method is based on the metafeatures of the processed data stream.

The processing procedure of 2PAC integrates the operation of two components: (a) prior probability estimator and (b) selected base classifier that supports incremental learning. The method transfers all available objects to both those components. The prior probability for the current data batch is first estimated, then, based on the estimated value and the support of the classifier towards specific classes, the method compensates the predictions by adjusting the predictions in favor of the minority class.

The operation of the method is presented in Algorithm 5. Its processing scheme depends on the three main hyperparameters:

- *base classifier* (Ψ) – the specified classifier will be incrementally trained and used to provide the probability of class predictions. The method should, therefore, allow for incremental training and as well provide the support function of classification task. The final decisions returned by 2PAC will base on the support function of the trained classifier.
- *prior probability estimation method* $(\mathcal{PE})$ – this module is used to calculate the estimated number of specific class occurrences and later combined with the decisions of the embedded classifier to compensate its bias toward majority class. Any module that estimates the prior probability of the data batch can be used.
- *threshold* (θ) – this parameter describes a value of estimated prior probability for a minority class to perform a prediction correction. With a maximal possible

Algorithm 5 Pseudocode of the *Prior Probability Assisted Classifier* method

```
Input:
    $\mathcal{DS} = \{\mathcal{DS}_1, \mathcal{DS}_2, \dots, \mathcal{DS}_k\}$ – data stream
                                                            ▷ Hyperparameters
    $\Psi$ – base classifier
    $\mathcal{PE}$ – prior probability estimator
    $\theta$ – threshold
                                                            ▷ Parameters
    $p_{min}$ – estimated prior probability of minority class
    $F_{min}$ – support of $\Psi$ towards minority class

 1: for all $\mathcal{DS}_k \in \mathcal{DS}$ do
 2:     if $k > 1$ then              ▷ Estimate prior probability and supports towards minority class
 3:         $p_{min} \leftarrow \mathcal{PE}(\mathcal{DS}_k)$
 4:         $F_{min} \leftarrow \Psi(\mathcal{DS}_k)$                ▷ Correction towards minority class
 5:         if $p_{min} < \theta$ then              ▷ Calculate number of correction samples
 6:             $cc \leftarrow |\mathcal{DS}_k| \times p_{min}$                 ▷ Conduct prediction correction
 7:             sort $DS_k$ by $F_{min}$
 8:             label last $cc$ samples of $DS_k$ as minority class
 9:         end if
10:     end if          ▷ Update classifier and prior probability estimator with current data chunk
11:     update $\Psi$ with $\mathcal{DS}_k$
12:     update $\mathcal{PE}$ with $\mathcal{DS}_k$
13: end for
```

value of $\theta = 50\%$, a correction is made for each data chunk, regardless of the estimated prior probability. Manipulating the hyperparameter allows the prediction to be corrected only in the event of a substantial imbalance being detected, leaving the original predictions otherwise.

For the first chunk of the data stream, only the update of the classifier and the prior probability estimator are performed (lines 11:12 in the pseudocode). In all other data chunks, the following steps are performed:

1. The estimation of prior probability for minority class is calculated based on samples from the current data chunk (line 3).
2. The classifier's supports towards minority class are calculated (line 4).
3. If the estimated prior probability p_{min} is lower than threshold θ, the value of the desired number of minority class samples is calculated (lines 5:6), followed by the compensation of classifiers predictions for the desired number of the minority class samples (lines 7:8).
4. After each chunk processing, the classifier and prior probability estimator are updated (lines 11:12).

4.2.1 The design of experiments

This subsection describes the data streams used in the research, the experimental setup, and the goals of specific experiments, aiming to thoroughly investigate the benefits and limitations of 2PAC approach. Since the method is directly dedicated to the data characterized with prior probability changes, the evaluation focuses on this type of data stream non-stationarity. In all experiments, the metric of balanced accuracy score was considered.

Data streams In order to reliably evaluate the proposed approach, three main types of imbalanced synthetic data streams were analyzed in the context of the data imbalance ratio:

- *Statically imbalanced streams* (SIS) – containing 10, 5 and 2.5% of minority class instances,
- *Continuous dynamically imbalanced streams* (CDIS) – with four drifts of prior probability, each described by a change amplitude of 75, 90, or 100%,

- *Discrete dynamically imbalanced streams* (DDIS) – with a standard deviation of 10% and the presence of minority instances at the level of 10, 5, and 2.5%.

In total, nine configurations describing a data imbalance were considered in those three taxonomic categories. The complete description of the data stream generator configuration is presented in Table 4.5, including the precise description of changes visible in prior probability.

All streams consisted of 500 data chunks, each containing 200 instances described by eight informative features. For the purpose of reliable experimental evaluation, each stream was replicated ten times with various random seeds.

Table 4.5: *The configuration of data stream generator in the experiments performed for* 2PAC *method. The first column presents the characteristics of the data, and the following one – the specified configuration.*

CHARACTERISTICS	CONFIGURATION
Dynamics of class imbalance	SIS, CDIS, DDIS
– Imbalance ratio in SIS	10, 5, and 2.5%
– Number of drifts in CDIS	4
– Imbalance amplitude in CDIS	75, 90, and 100%
– Imbalance ratio in DDIS	10, 5, and 2.5%
Number of chunks	500
Chunk size	200
Number of features	8 (100% informative)
Replications	10

Hyperparameter optimization The first experiment aimed to optimize the method's hyperparameters for five baseline classification approaches. The experiments used independent classifiers capable of incremental learning and classifiers integrated into a simple ensemble, making them suitable for an incremental learning environment. Table 4.6 presents the evaluated classifiers and their configuration.

Table 4.6: *The table presenting the classification approaches taken into account in the comparative experiment. The first columns present the method's acronym and its name, and the final one – the specified configuration of a method.*

METHOD		CONFIGURATION
MLP	*Multilayer Perceptron*	a single hidden layer consisting of 10 neurons, the default *ReLU* activation function, *Adam* solver; the incremental learning scenario applied a single iteration (epoch) per data batch
GNB	*Gaussian Naive Bayes*	the default variance smoothing of 1e-9
KNN	*k-Nearest Neighbors*	SEA with KNN as the base classifier; 10 estimators in an ensemble; number of neighbors set to 5; uniform weights of neighbors
SVM	*Support Vector Machine* [43]	SEA with SVM as the base classifier; 10 estimators in an ensemble; SVM with RBF kernel and the regularization of $C = 1$
HTC	*Hoeffding Tree Classifier* [103]	the default Information Gain split criterion, the default number of instances a leaf should observe between split attempts equal to 200

Three of the evaluated classifiers (namely MLP, GNB and HTC) allowed for the incremental training by default. For the remaining two methods – KNN and SVM classifiers –*Streaming*

Ensemble Algorithm (SEA) [216] was used in order to allow the incremental processing of data. The implementation of the original SEA ensemble approach has been slightly modified in order to process highly imbalanced data streams, characterized by the absence of minority class objects in the first chunk of the stream. In such a case, additional label noise is introduced, forcing the presence of at least one representative of the minority class.

In this experiment, three strategies of prior probability estimation $\mathcal{PE}$ were evaluated:

- the baseline approach, which assigns the current data chunk the same prior probability as in the previous data batch (PREV),
- the estimated prior probability calculated as a mean prior probability from all previously processed data batches (MEAN),
- *Dynamic Statistical Concept Analysis* (DSCA) [124] – the original approach which estimates the current prior probability based on statistical metafeatures of a data chunks, integrated using the regression mechanism.

All prior probability estimation methods utilize the metafeatures computed for the data chunks. The simple approaches – PREV and MEAN – directly use the prior probability from the past chunks. Meanwhile, the DSCA approach uses the mean values of features and their standard deviation as metafeatures, and the number of class representatives as metatargets, later integrated to estimate the prior probability. This mechanism makes it the most complex out of all the evaluated approaches.

The experiments considered a range of values from 1% to 50% for the *threshold*. In the case of a value of 50%, the correction of the predictions returned by the classifier is made for each data batch, even when the estimation method does not indicate any imbalance. A low value of the *threshold* allows the correction of the classifier's prediction only when a substantial imbalance is detected.

A significant improvement in the classification quality is expected with low values of the θ, which remains at the static level as the value of this hyperparameter increases. In the case of significant estimation errors, which may result from the method not adapted to the type of stream, with high values of this hyperparameter, the classification quality may decline. This will be a result of unnecessary changes in the classifier's initially correct decisions.

Comparison of evaluated methods The second experiment aimed to compare the proposed 2PAC method with the recognition quality of the classifiers without the support offered by prior probability estimates and the compensation mechanism.

The estimation quality may significantly impact the possibility of correcting the classifier's output. When determining the level of class imbalance with a significant error, the 2PAC method would be degenerating the classifier's performance. The preliminary studies suggest that various prior probability estimation methods could achieve the most accurate prior probability estimates, depending on the stream type. Assuming the sub-optimal selection of such a method and effective prior probability estimation, the application of the 2PAC method should positively impact the classification quality compared to the base classifier, especially in the case of a significant imbalance.

4.2.2 Experimental evaluation

This subsection presents the effects of the experiments performed on 2PAC method, along with the analysis of their results. After selecting the method's hyperparameters and configuration in the first experiment, the following one evaluated the proposed solution compared to the baselines.

Hyperparameter optimization The first experiment aimed to analyze the method's hyperparameters and find their suboptimal values. As a supplement, the errors in prior probability estimation of all tested methods were evaluated. They are presented in Figure 4.7 as the accumulated average error values for the assessed prior probability estimation methods. The results for SIS are shown in the first row, followed by the results for CDIS and DDIS. The specific colors indicate the accumulated error values of various prior probability estimation methods.

In *statically imbalanced streams* (SIS), it is noticeable that all tested methods have a similar error level. At the beginning of the stream processing time, the DSCA method obtains the largest estimation errors, which later decrease over time. The lower the proportion of the minority class, the lower the error values.

The second row shows the errors of *continuous dynamically imbalanced streams*. It can be noticed that the MEAN method shows significant estimation errors for this type of stream. Such errors result from the type of imbalance changes in these streams. When the minority class transitions to the majority class, the mean value fluctuates around 50%, giving almost no information about the actual prior probability. The moments of a significant decrease in estimation error occur for short periods when the problem becomes balanced. It can also be seen that the error values of the PREV method increase on intervals of greater dynamics of changes in the class proportions in the stream.

The last row presents the errors for processing *discrete dynamically imbalanced streams*. The PREV method achieves the highest error values, as these streams are characterized

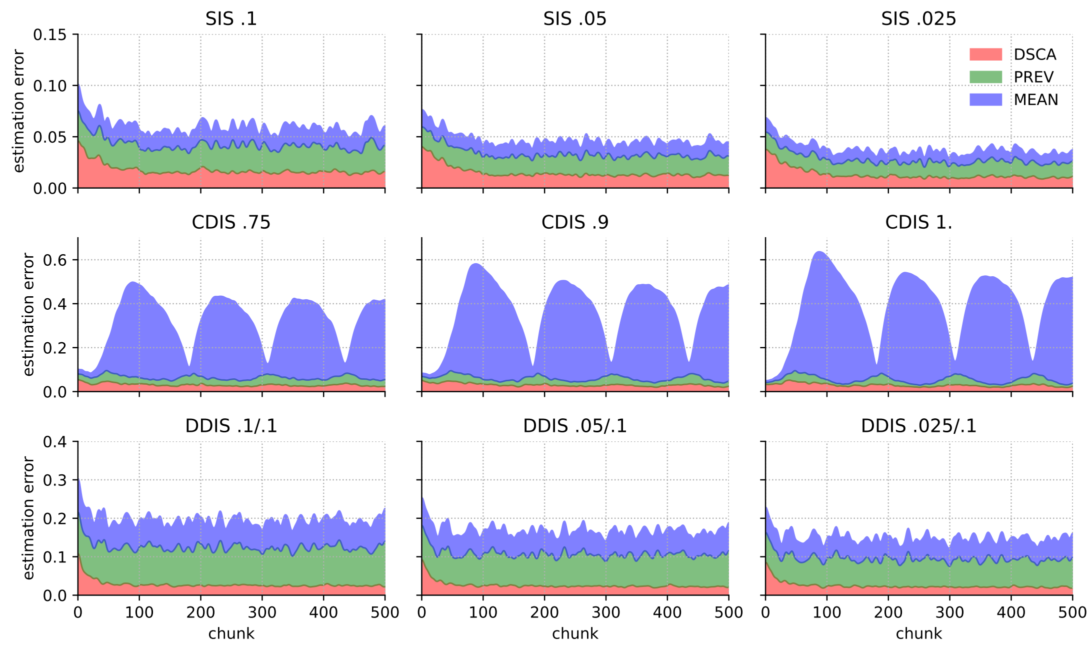

Figure 4.7: *The accumulated mean error of prior probability estimation for evaluated streams. The rows show different types of changes in prior probability in the data stream, and the columns the degree of such changes. The colors indicate the errors obtained by the specific estimation methods.*

by significant differences in prior probability in adjacent data chunks. As with *statically imbalanced data streams*, the error of DSCA is more significant at the beginning of processing and decreases over time. The DSCA method achieves the most promising results for *dynamically imbalanced streams*.

Figure 4.8 shows the results of the threshold hyperparameter selection performed in the first experiment for all base classifiers and all prior probability estimation methods. As the observations for different prior probability dynamics were consistent, the results were averaged for all replications, and the levels of imbalance change (imbalance ratio for SIS and DDIS or change amplitude for CDIS). The columns of the figure show the results for different types of imbalance change, and the rows – for different base classifiers. Each color of the plot represents a specific prior probability estimation method.

As expected, in some cases, the classification quality increases with a low θ, and the value remains at a high level with the further increase of hyperparameter values. This occurrence can be seen especially in the case of SIS and DDIS.

The exceptions are observed in CDIS streams where, in the case of the MEAN estimation method, the results either decrease or increase with a significant value of the θ. The decrease in the results is related to the significant errors in the prior probability estimation, also visible in Figure 4.7 for this stream type.

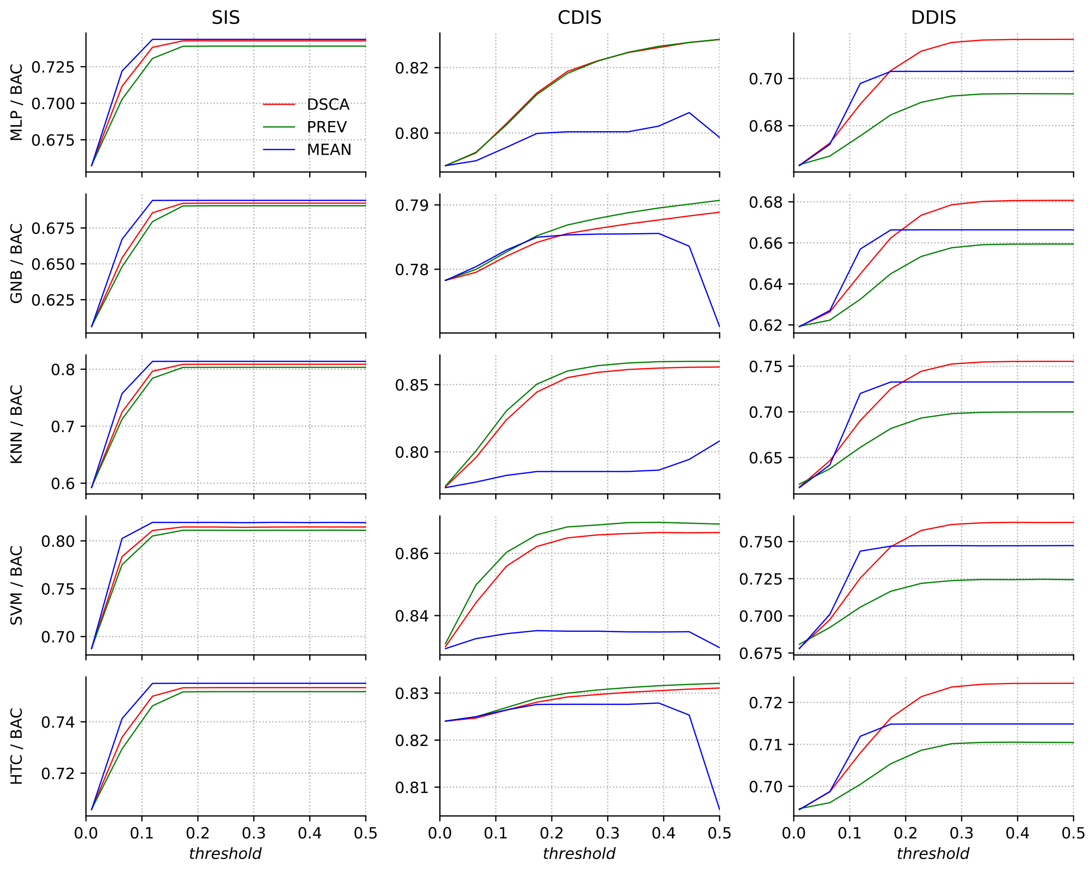

Figure 4.8: *The results of hyperparameter selection experiment. The balanced accuracy score was averaged for the prior probability change type, presented in columns. The rows present the classification quality for evaluated baseline classification methods. The colors of the plots indicate the prior probability estimation method.*

In the streams, where the initial minority class at some point in time becomes the majority of observed instances, considering the periods of imbalance are constant, the prior estimate oscillates around 50%. Setting a high *threshold*, i.e., always taking into account the estimation value during prediction, forces a balanced prediction of an imbalanced data stream, which in most cases hurts the classification quality. For some base classifiers, in streams characterized by a large amplitude of changes, the improvement of the results with high values of the θ results from the correction of the classifier's prediction at the moments when the actual prior probability leans towards the value determined by the $\mathcal{PE}$, which in the case of the MEAN method for the presented streams is around 50%.

The PREV and DSCA methods, with the increase in the value of the θ, achieve better results. Thus, it can be concluded that the correction of the classifier prediction is beneficial even in the case of chunks with slight differences in the number of class instances.

In SIS, the results for each of the evaluated base classifiers were similar and satisfactory for each of the prior probability estimation methods. In CDIS, the PREV and DSCA methods perform best, with a slight advantage of the PREV method for streams with a large amplitude of imbalance changes. For DDIS, the most promising estimation method is DSCA, and the least improvement is provided by the PREV method.

For the analyzed types of streams, it was decided to choose different values of θ hyperparameter, depending on the method of prior probability prediction, for the MEAN $\theta = 28\%$, for PREV and DSCA $\theta = 50\%$.

Comparison of evaluated methods The second experiment analyzed the impact of the proposed 2PAC method on the classification quality. The operation of the method, optimized for each type of imbalance dynamics, was compared with the operation of the baseline classification approach without prediction corrections.

Figure 4.9 shows the averaged balanced accuracy score values for streams characterized by the highest imbalance ratio or the highest amplitude of changes in the prior probability of each of the analyzed stream types. The subsequent rows show different types of streams, respectively SIS, CDIS and DDIS. All considered prior probability estimators were integrated with each of the tested base classifiers – presented in the columns – using the 2PAC method. Additionally, the black line marks the balanced accuracy score of the baseline classification method.

Additionally, Figure 4.10 shows the cumulative results to enhance the interpretation of the results. The value presented in the plots is equal to the average result from all

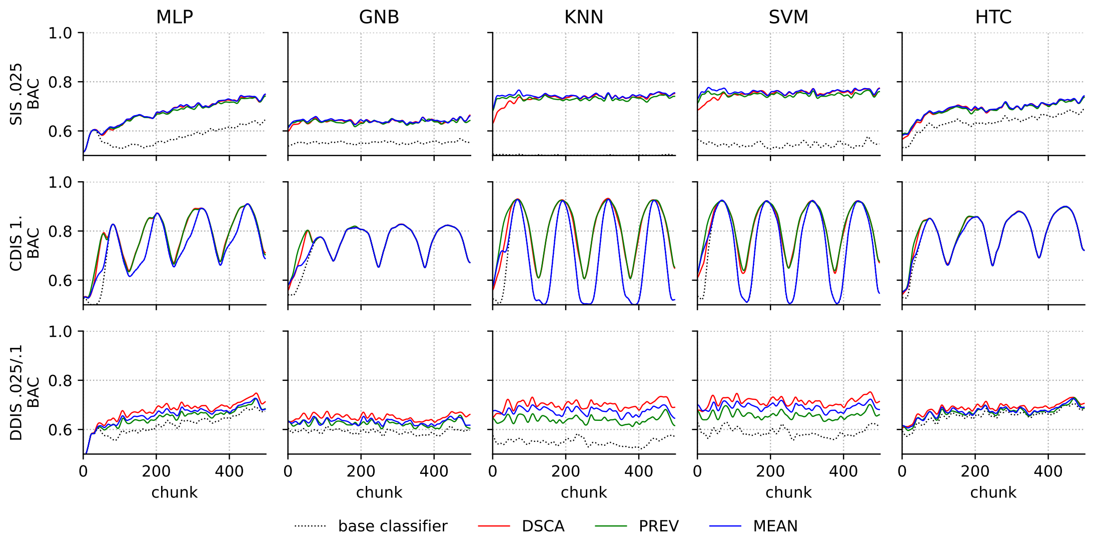

Figure 4.9: *The results of a comparison experiment for* 2PAC *method. The plots show the average balanced accuracy score for evaluated classifiers (shown in columns) and the types of streams (shown in rows). The prior probability estimation method integrated with* 2PAC *is identified with the specific colors of plots.*

chunks processed so far. This allows for immediate identification of the method with the highest average classification quality – the one with the highest value for the last chunk of the processed stream.

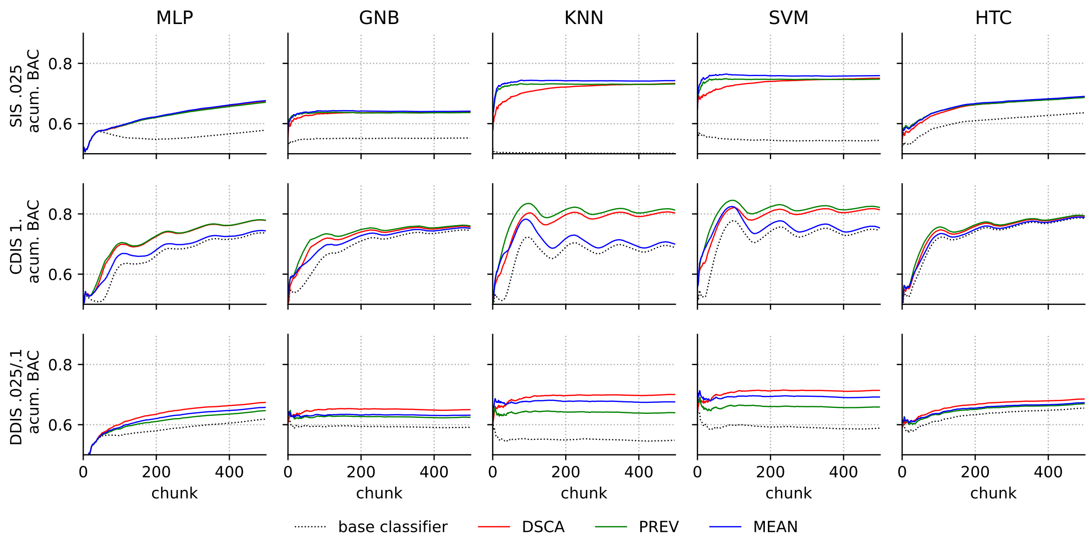

Figure 4.10: *The results of a comparison experiment for* 2PAC *method. The plots show the cumulative balanced accuracy score for evaluated classifiers (shown in columns) and the types of streams (shown in rows).*

The results of all evaluated imbalanced streams show an improvement in the classification quality when using 2PAC. The extent of improvement of using the method compared to the baseline depends on the combination of classification and estimation methods employed.

The most significant benefits of using the method can be seen in the case of the KNN and SVM classifiers. For SIS, in the case of the two methods mentioned above, slight differences in balanced accuracy can be seen in employed estimation methods, favoring the MEAN and PREV methods, especially in the initial stage of stream processing. In the case of the other classifiers, the differences between prior probability estimation methods are not clearly noticeable.

In CDIS streams, it is possible to observe how a given base method reacts to changes in the problem imbalance ratio by the quality of the baseline classifier itself. Those continuous changes have a particularly negative impact on the KNN and SVM classifiers – the value of the accumulative average quality of the classification decreases rapidly in moments of significant imbalance. For the MEAN estimation method, benefits are visible in the initial processing stage. However, the classification quality becomes almost equivalent to the baseline classifier over time. In the case of the GNB, KNN, and SVM classifiers, the most promising results are obtained by the PREV estimation method.

The difference is not noticeable in the case of MLP and HTC classifiers. Still, PREV estimation method seems to be best suited for this type of imbalance dynamics.

For the DDIS streams, presented in the last row of figures, the most promising classification results are achieved by the base method integrated with the DSCA prior probability estimator. The quality improvement is least noticeable in the case of the HTC classifier.

As an effect of previous analyses, the 2PAC method was integrated with the prior probability estimation method, showing the best classification quality and the smallest estimation errors. For SIS streams, the MEAN method was selected, while for CDIS streams with smooth changes in the imbalance ratio, the PREV method was selected. For DDIS streams, the DSCA method was characterized by the smallest errors and was selected as the default approach.

The results of classification quality for all examined streams are presented in Table 4.7, supplemented with the Student's T-test results for the significance level *alpha* = 5%. The first column of the table describes the type of imbalance and its dynamics in the considered streams. The subsequent columns describe the results for the five basic methods and for the methods with prediction compensation using the proposed 2PAC approach.

Table 4.7: *The results of the experiments conducted to evaluate* 2PAC *approach. The columns present the evaluated methods – with the ones integrated with* 2PAC *highlighted in red – and the rows show the types of data streams regarding class imbalance.*

STREAM	*MLP*	*2PAC-MLP*	*GNB*	*2PAC-GNB*	*KNN*	*2PAC-KNN*	*SVM*	*2PAC-SVM*	*HTC*	*2PAC-HTC*
	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
SIS .1	0.744	0.809	0.665	0.746	0.708	**0.876**	0.822	**0.871**	0.777	0.817
	2 4	0 2-4 8	—	2 4	—	**0-4 6 8 9**	0 2-4 8	**0-4 6 8 9**	0 2-4	0 2-4 8
	—	x	—	x	—	x	—	x	—	x
SIS .05	0.650	0.746	0.601	0.695	0.567	**0.823**	0.696	**0.827**	0.705	0.758
	2 4	0 2-4 6 8	—	0 2 4	—	**0-4 6 8 9**	0 2 4	**0-4 6 8 9**	0 2 4	0 2-4 6 8
	—	x	—	x	—	x	—	x	—	x
SIS .025	0.578	0.676	0.552	0.641	0.501	0.743	0.544	**0.759**	0.636	0.690
	4 6	0 2-4 6 8	4	0 2 4 6	—	0-4 6 8 9	4	**all**	0 2 4 6	0 2-4 6 8
	—	x	—	x	—	x	—	x	—	x
CDIS .75	0.842	0.868	0.804	0.811	0.853	**0.904**	0.892	**0.905**	0.856	0.861
	2 3	0 2 3	—	2	2 3	**0-4 6 8 9**	0-4 8 9	**0-4 6 8 9**	0 2 3	0 2 3 8
	—	x	—	x	—	x	—	x	—	x
CDIS .9	0.792	0.829	0.785	0.796	0.779	**0.878**	0.851	**0.882**	0.831	0.839
	—	0 2-4	—	2	—	**0-4 6 8 9**	0-4 8	**0-4 6 8 9**	0 2-4	0 2-4 8
	—	x	—	x	—	x	—	x	—	x
CDIS 1.	0.736	0.769	0.745	0.757	0.689	0.81	0.745	**0.821**	0.785	0.792
	4	0 2 4 6	4	0 2 4	—	0-4 6 8 9	4	**all**	0-4 6	0-4 6 8
	—	x	—	x	—	x	—	x	—	x
DDIS .1/.1	0.720	0.768	0.656	0.719	0.710	**0.819**	0.783	**0.820**	0.746	0.774
	2	0 2-4 8	—	2	2	**0-4 6 8 9**	0 2-4 8	**0-4 6 8 9**	0 2-4	0 2-4 8
	—	x	—	x	—	x	—	x	—	x
DDIS .05/.1	0.650	0.708	0.610	0.673	0.593	0.747	0.663	**0.755**	0.683	0.714
	2 4	0 2-4 6 8	—	0 2 4		0-4 6 8 9	2 4	**all**	0 2 4 6	0 2-4 6 8
	—	x	—	x	—	x	—	x	—	x
DDIS .025/.1	0.619	0.674	0.591	0.650	0.548	0.700	0.589	**0.714**	0.655	0.685
	2 4 6	0 2-4 6 8	4	0 2 4 6	—	0-4 6 8 9	4	**all**	0 2 4 6	0 2-4 6 8
	—	x	—	x	—	x	—	x	—	x

The methods integrated with 2PAC are highlighted in red. The table presents the averaged balanced accuracy score within the replications and the results of statistical tests. The indexes below the average classification quality indicate the references from which the method is statistically significantly better. Additionally, the markers indicate the statistically significantly better method within a pair: baseline compared to the method integrated with 2PAC. The best results for each stream type are marked in bold.

The results show that globally – across all the studied methods – the best quality, regardless of the prior probability dynamics, is achieved by the KNN and SVM methods integrated with 2PAC, as the baseline classifiers, KNN and SVM are the only ones embedded in the SEA ensemble classification method. Such results show the advantage of the ensemble approaches in the classification of data streams in relation to the use of single baseline classifiers, which was already noticed in the literature [80]. The table also shows that for each of the studied stream types, the classifier integrated with the 2PAC method achieves statistically significantly better results than the baseline method alone.

*The overall results show that the use of prior probability-based correction allows for achieving statistically significantly better classification quality in data streams of various types and degrees of change in the level of a class imbalance. The employed prior probability estimation methods used basic metafeatures related directly to the class imbalance from the past data chunks (*PREV *and* MEAN *approaches), as well as a complex method of Dynamic Statistical Concept Analysis (*DSCA*), analyzing and integrating the statistical metafeatures of data chunks.*

4.3 Certainty-based Architecture Selection Framework

This section describes the *Certainty-based Architecture Selection Framework* (CAS) designed for processing data streams with time-varying concept difficulty. Such a specific concept drift can be defined as a change in the *domain* of processing [224], where the considered classification problem, including the definition and semantics of the recognized classes, remains constant. At the same time, the instances are characterized by varying difficulty over time.

This particular method was designed with a real-life *TinyML* use case in mind. In this vivid research area the low processing time and, therefore, low complexity of methods become especially important due to limited computational resources of embedded systems [57]. The dynamic selection of appropriate neural network architecture with the proposed CAS framework can allow the modulation of a trade-off between the computational complexity of methods and the classification accuracy of a system. Keeping

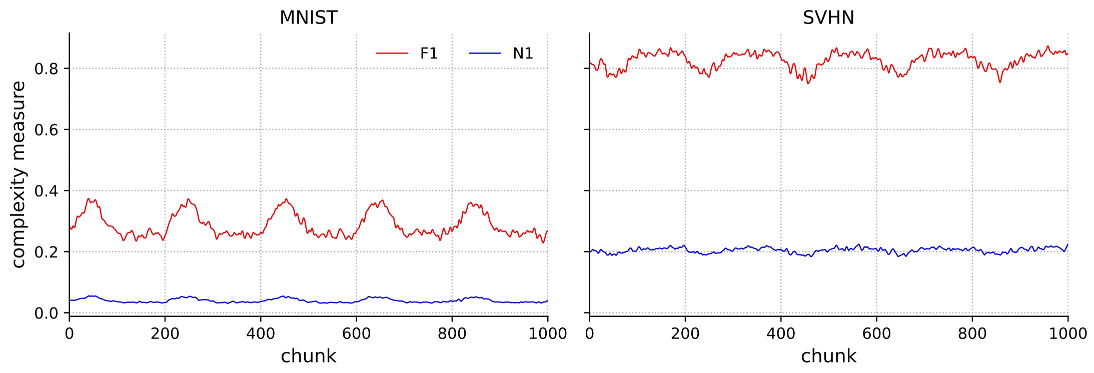

Figure 4.11: *The plots describing the F1 and N1 complexity measures averaged over results from the one-versus-all strategy for examined data streams after feature extraction. The left side presents the measures for* MNIST *dataset and the right for* SVHN *[174]. The five cycles of difficulty variability are visible over the stream course for both streams, more significantly exhibited in the $F1$ measure.*

this real-world application in mind, the research uses the metrics dedicated to *TinyML*, measured on a hardware device, to estimate the described trade-off.

Figure 4.11 aims to present the type of data variability considered in this processing scenario. It shows the difficulty of data streams created from MNIST [51] and SVHN static *computer vision* datasets. In the figure, the complexity of the classification task [154] is calculated according to averaged measures of $F1$ and $N1$ from the complexity metafeature category, described in Subsection 2.1.2. The measures were calculated within data chunks on the samples transformed to 10 features with PCA. The generated streams were subjected to feature reduction and binarization with a one-versus-all (OVA) strategy to allow the calculation of the complexity measures. Such measures are supposed to highlight the objective measures (unbiased in terms of the employed classification model), showing the time-varying difficulty of the solved task.

The plots presented in the figure highlight the changes in problem complexity occurring in the stream. By adapting to those changes on an architecture selection level – by selecting simpler architecture for a simpler problem and a more complex one for a difficult concept – a classification accuracy should potentially be able to remain constant while altering the computational or time complexity of the solution.

The presented framework aims to allow for such selection of the CNN architecture. The estimation of problem difficulty – and thus the selection of the architecture – is based on the analysis of the classifier *certainty*. The integrated outputs from the static models are used as data stream metafeatures, where each data chunk is described by a deterministic *mean predictive support* value – a metafeature that estimates the *certainty* of the classifier. The complete operation of the framework is described in Algorithm 6.

Algorithm 6 Pseudocode of the *Certainty-based Architecture Selection Framework*

```
Input:
    $\mathcal{DS} = \{\mathcal{DS}_1, \mathcal{DS}_2, \ldots, \mathcal{DS}_k\}$ – data stream
                                                        ▷ Hyperparameters
    $\Pi = \{\Psi_1, \Psi_2, \ldots, \Psi_e\}$ – pool of NN architectures
    $\Theta = \{\theta_1, \theta_2, \ldots, \theta_e\}$ – list of support thresholds
    $s$ – number of chunks in given range to switch
    $s_c$ – counter of chunks in given range to switch
                                                        ▷ Parameters and functions
    $c$ – index of selected architecture
    $\mathcal{S}$ – calculate softmax
    $\mathcal{MPS}$ – calculate mean predictive support

 1: Initialize $c$ as $floor(\frac{e}{2})$
 2: for all $\mathcal{DS}_k \in \mathcal{DS}$ do          ▷ Forward pass of currently selected architecture
 3:     $p \leftarrow \mathcal{S}(\text{forward } \Psi_c(\mathcal{DS}_k))$
 4:     $\rho \leftarrow \mathcal{MPS}(p)$
 5:     $c_{temp} \leftarrow$ index of minimum $\theta$ from $\Theta$ exceeding $\rho$
 6:     if $c_{temp} \neq c$ then
 7:         if $s_c = s$ then
 8:             if $c_{temp} > c$ then               ▷ Select more complex architecture
 9:                 $c \leftarrow c + 1$
10:             else                                 ▷ Select less complex architecture
11:                 $c \leftarrow c - 1$
12:             end if
13:             $s_c \leftarrow 0$
14:             else                                 ▷ Increment the switch counter
15:             $s_c \leftarrow s_c + 1$
16:         end if
17:         else
18:         $s_c \leftarrow 0$
19:     end if                                       ▷ Utilize prediction probabilities p
20:     return predictions for $\mathcal{DS}_k$ as $argmax(p)$
21: end for
```

The framework selects a single NN model to be used in the current data chunk from a pool of architectures already optimized for the problem under consideration. All architectures should be trained until they reach a specified *mean predictive support* (MPS), denoted by ρ so that this measure can serve as a *proxy* in estimating the difficulty of a chunk of data in the stream. The MPS is calculated based on the output from the last layer of the considered network architecture – as presented in lines 3:4 of the pseudocode. Only the network outputs for the one-hot-encoded labels that would lead to a prediction in favor of a predicted class (i.e., the one with the highest prediction probability) are considered. Then, the average support value within the data chunk is calculated. The difficulty of the classification task being solved (related to the number of errors) is intuitively dependent on the *certainty* of prediction – the higher the model certainty is, the more a considered instance should fit into the distribution of a given class.

During preliminary research, it was also observed that despite optimizing the weights of neural networks to a specific MPS level, the number of errors made by the classifier, and thus the classification accuracy, varies. Therefore, these observations allow proposing a solution in which more computationally expensive network architectures are reserved for the stream domain, where models show lower *certainty*. While predictions are made with high certainty, a less computationally expensive architecture is used for inference.

In addition to the pool of e trained architectures $\Pi = \{\Psi_1, \Psi_2, \dots, \Psi_e\}$ (sorted from the least to the most complex), the method adopts a list of thresholds $\Theta = \{\theta_1, \theta_2, \dots, \theta_e\}$ (from the largest to the smallest) determining the MPS value from which individual architectures are used. Since switching the model on hardware may involve a time delay or cause temporal instability of the system, the s hyperparameter has been implemented, specifying how many data chunks of a processed stream must be outside the current MPS range in order to switch to a more or less complex architecture. Such switching exchanges the model by no more than one step for detected change in certainty. At the beginning of processing, the architecture in the middle of the pool Π is selected, as presented in line 1 of the pseudocode.

For each data chunk, the following steps are performed:

1. The *softmax* function is calculated with the forward pass of the currently selected architecture (line 3).
2. MPS ρ is calculated based on the supports for all classes and all objects in the batch (line 4).
3. The value of c_{temp} is then determined – the smallest threshold index from the list Θ that exceeds ρ (line 5).
4. Before switching to the architecture with index c_{temp}, the method checks whether a sufficient number of s chunks have passed since the first threshold crossing for the current c architecture (line 7).
5. If the last s of chunks indicated the need to change the architecture, it is switched (lines 8-11).
6. If the number of s_c was not achieved, the counter is incremented (line 15), or if the switch was not requested, the counter is set to zero (line 18).
7. At the final stage, the predictions are obtained from the raw output from *softmax* function, determined at the beginning of processing (line 20).

The choice of *certainty* as the architecture switching criterion results from the low computational overhead associated with estimating the difficulty of the recognized concept. After determining the MPS based on the *softmax* output, it is possible to calculate the predictions themselves. The proposed framework, therefore, allows for smooth trade-off modulation between the inference time and the *accuracy* without significantly increasing the computational complexity of the classification task, which was seen for using the classic *complexity* metafeatures.

4.3.1 The design of experiments

This subsection presents the generated data streams, describes the research environment and the planned experiments along with their goals. Since the CAS approach is the only one presented in this dissertation that aims to directly minimize the processing time and computational complexity with minimal effect on the classification accuracy, this subsection describes the *Time to Accuracy Loss* metric, introduced to support the evaluation of the proposed solution.

Data Streams The research was performed on semi-synthetic data streams, where two popular *computer vision* datasets were arranged into an ordered data stream characterizing the non-stationary problem difficulty.

Table 4.8: *The table describing the data configuration for the generated semi-synthetic data streams. The first column describes the characteristics of the data streams, and the second one their selected values.*

CHARACTERISTICS	CONFIGURATION
Origin dataset	MNIST, SVHN
Original samples	60 000 (MNIST), 73 000 (SVHN)
Number of chunks	1000
Chunk size	50, 150, 300, 500
Complexity cycles	3, 5, 10, 25
Difficulty change type	*incremental*
Replications	5

Two real-world datasets used for this purpose were (*a*) the simpler MNIST, which shows monochromatic handwritten digits, and (*b*) the more challenging SVHN, which shows color images of digits describing house numbers. Both datasets describe a balanced, multiclass classification of 10 object categories. The streams were synthesized from the static datasets using an original generation method. The characteristics of this data are described in Table 4.8.

The stream for the MNIST set was developed using 60 thousand original objects and for the SVHN set using 73 thousand original objects. Each of the considered data streams consisted of one thousand data chunks. Various chunk sizes – from 50 to 500 – and numbers of difficulty change cycles – from 3 to 25 cycles – were considered in the experiments. All concept changes were of *incremental* type, presenting a smooth dynamics of change. The generation was replicated five times for all data stream configurations.

Evaluation metrics The research was carried out on 10-class balanced data streams. Therefore, classification quality was assessed using an *accuracy* metric. In addition to assessing the classification quality, individual architectures' inference time measured in *Multiplier ACCumulator* MACC and *latency* was examined [95]. The MACC is an auxiliary measure typical for TinyML applications, that describes the number of operations of multiplying two values and adding a third one, while *latency* is a direct measure describing the inference time.

Additionally, to facilitate the assessment of the trade-off between classification quality and operation time, the *Time to Accuracy Loss* (TTAL) measure was proposed, which is described in Equation (4.2).

$$TTAL = \frac{R_{macc} \cdot R_{latency}}{R_{acc}} \tag{4.2}$$

This is a relative criterion intended to compare the result of the proposed solution with the reference method, which is assumed to have a higher classification quality and a higher inference time expressed in MACC and *latency*. Calculating TTAL measure requires calculating the *relative loss measures* R_m, described in Equation (4.3), where m stands for the base metrics, i.e., the MACC, *latency*, and *accuracy*.

$$R_m = \frac{m_{reference} - m_{considered}}{m_{reference}} \tag{4.3}$$

Therefore, to calculate the value of TTAL, first the relative values of R_{macc}, $R_{latency}$, and R_{acc} are required, calculated according to reference and obtained MACC, *latency*, and *accuracy*, respectively. The TTAL measure returns high values for the low *accuracy* loss in relation to the time loss. Therefore, the goal is to maximize this criterion.

Neural Architecture Selection The first experiment was intended to select a pool of NN architectures considered for further processing using the CAS framework. A total number of 14 architectures were examined, from the simplest – containing only *fully connected* layers – to the more complex – containing multiple *convolutional* layers. For the proposed framework, a specific set of considered architectures should be predefined, where the more time-demanding solution achieves a better classification quality. A two-criteria optimization task can be formulated, where one criterion is the inference time and the other is the recognition quality.

The selected NN architectures, suitable for the CAS framework, belong to a *Pareto set* of this two-criteria optimization task [143]. Such a set describes the non-dominated solutions – where the results of specific approaches describe the different optimal trade-offs among the objectives considered in the multi-criteria optimization task [148]. This experiment seeks for the NN architectures that offer the best classification quality in a given processing time – and conversely – that offer the shortest processing time for a specific classification accuracy.

The experiment was carried out on static data, which was used for the training in the following experiments. The dataset was divided into five folds in stratified cross-validation.

Architecture training was stopped at epoch 250 or when the MPS for the training set exceeded a given threshold. The average results of inference time and classification quality were analyzed. The experiment aimed to select five ordered architectures from the *Pareto set* that meet the assumption that a higher-accuracy model is more computationally expensive.

The operation of 14 neural network architectures was tested, which were divided into four families:

- *Fully Connected* (FC) – three architectures without convolutional layers with 2, 3, and 4 fully connected linear layers.
- *One Convolutional Layer* (CNN1) – four architectures with one convolutional layer, with depths of 5, 10, 15, and 20, respectively.
- *Two Convolutional Layers* (CNN2) – five architectures with two convolutional layers with increasing filter depth.
- *Three Convolutional Layers* (CNN3) – two architectures with three convolutional layers and increasing filter depth.

In each neural network architecture, a ReLU activation function was used, along with max-pooling layers of size 2x2 and a kernel size of five in the case of CNN. In the first experiment, the inference time of a specific architecture was measured in seconds on a desktop computer for the entire testing set of samples.

Advantage Estimation Experiment The second experiment aimed to verify whether the varying difficulty of the data affects the *certainty* of the classifier in the decision process and whether changing the classifier to a more complex one in the case of greater difficulty leads to an increase in classification quality. Based on this experiment's results, it was possible to estimate the decrease in average classification quality compared to the most complex architecture.

All considered network architectures were trained on an extracted static set to a specified level of MPS of 0.95 for the MNIST set and 0.9 for the SVHN set, or if such a value was not achieved, to a maximum of 250 epochs in the case of the MNIST set and 750 epochs in the case of the SVHN set. Those MPS thresholds were selected based on preliminary research according to the possibility of architectures to achieve a specified support level. *Stochastic Gradient Descend* and the *Cross Entropy* loss function were used for optimization.

A total of ten thousand objects were used to train architectures for MNIST, while for SVHN, there were 26 thousand objects in the training set. The weights were not fine-tuned during the stream processing – following the *Periodic holdout* processing protocol. This was motivated by two factors: (a) the decision to use the model's outputs as a deterministic metafeature that allows estimating the difficulty of the data, and (b) the operation of a semi-synthetic generator, that samples the data with replacement – and therefore, there is a possibility of sample duplicates in the stream. The model evaluation on samples already used to train the model would not allow for a reliable assessment of the recognition quality. While the incremental update of the model's parameters is skipped, the method offers the flexibility necessary for the non-stationary data stream processing by dynamic architecture switching.

As the application of hardware solutions is not among the main objectives of this dissertation, the detailed setup of the device used to estimate the time metrics was skipped. The metrics values obtained on the device experiment were used to estimate the benefits of employing the proposed framework in a real-life setting.

4.3.2 Experimental evaluation

This section presents and analyses the results of experiments performed on the proposed CAS approach. The first experiment aimed to select the appropriate components of the framework. The second one evaluated the outcomes of the solution by using the proposed TTAL measure – highlighting the benefits in processing time while marginally limiting the classification accuracy.

Neural Architecture Selection The first experiment evaluated 14 neural network architectures in the environment of two baseline datasets used in the research. The two considered components were the quality of classification and the inference time of a specific architecture.

Figure 4.12 shows the experiment's results for the MNIST and SVHN datasets. The points indicate the average results for the considered architectures. The solutions selected for the pool used in the framework – belonging to the *Pareto set* – are marked in color. The average classification quality is marked on the horizontal axis, and the inference time for the entire test set in seconds is marked on the vertical axis. The type of a marker indicates the model family – FC with circles, CNN1 with crosses, CNN2 with diamonds, and CNN3 with squares.

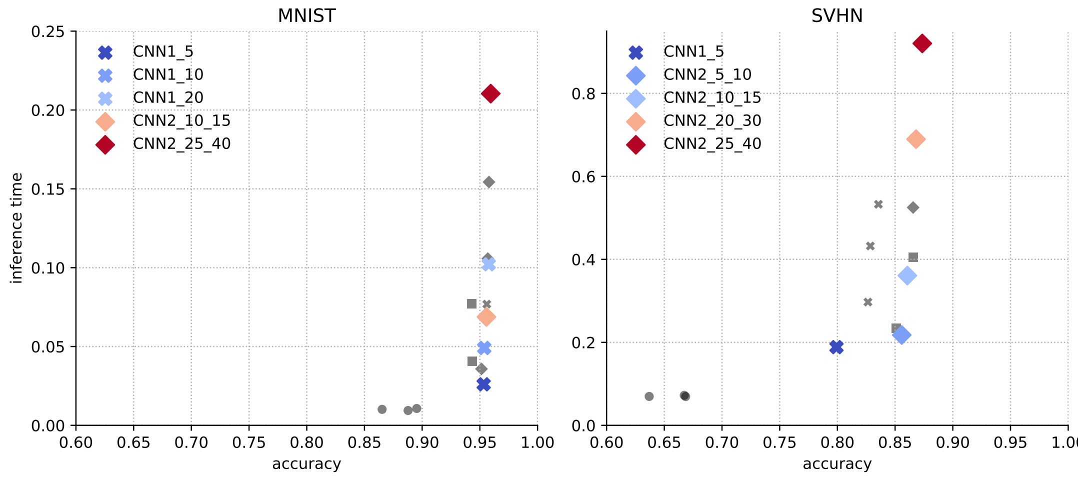

Figure 4.12: *The results of the first experiment, presenting the accuracy of the evaluated models (horizontal axis) and their inference time (vertical axis). The left side presents results for* MNIST *dataset and the right side for* SVHN*. The solutions selected as suitable for the presented framework are marked with colors and described in the legend.*

It can be easily noticed that the simplest architectures without convolutional layers (FC) are characterized by the lowest inference time but achieve very low classification quality. Therefore, they were not considered when selecting architectures for further experiments.

Architectures with one convolutional layer have performed relatively well in the MNIST problem, achieving satisfactory recognition quality with low inference time. Architectures with two convolutional layers performed best in terms of quality, but their inference time was the highest. Architectures with three layers did not perform well in this problem, which may be related to the relative simplicity of the task being solved. In the SVHN problem, CNN1 architectures achieved significantly lower quality results than CNN2 and CNN3. Architectures with two convolutional layers dominate the *Pareto set* for this dataset.

Advantage Estimation Experiment The following experiment was performed directly for the type of data under consideration – the streams with time-varying difficulty. The main goal of this experiment was to determine the quality of classification of individual architectures and the proposed solution, as well as to verify the ability to determine difficulty based on the designed MPS metafeature, describing the classifiers' certainty.

Figure 4.13 shows the averaged classification quality over time for selected streams generated from the MNIST and SVHN datasets. The rows of each figure represent a different number of cycles of difficulty changes – from three in the first row to 25 in the last one. The presented figures describe streams with a chunk size of 300 samples. The quality achieved by subsequent architectures is marked in blue. Light color indicates the least

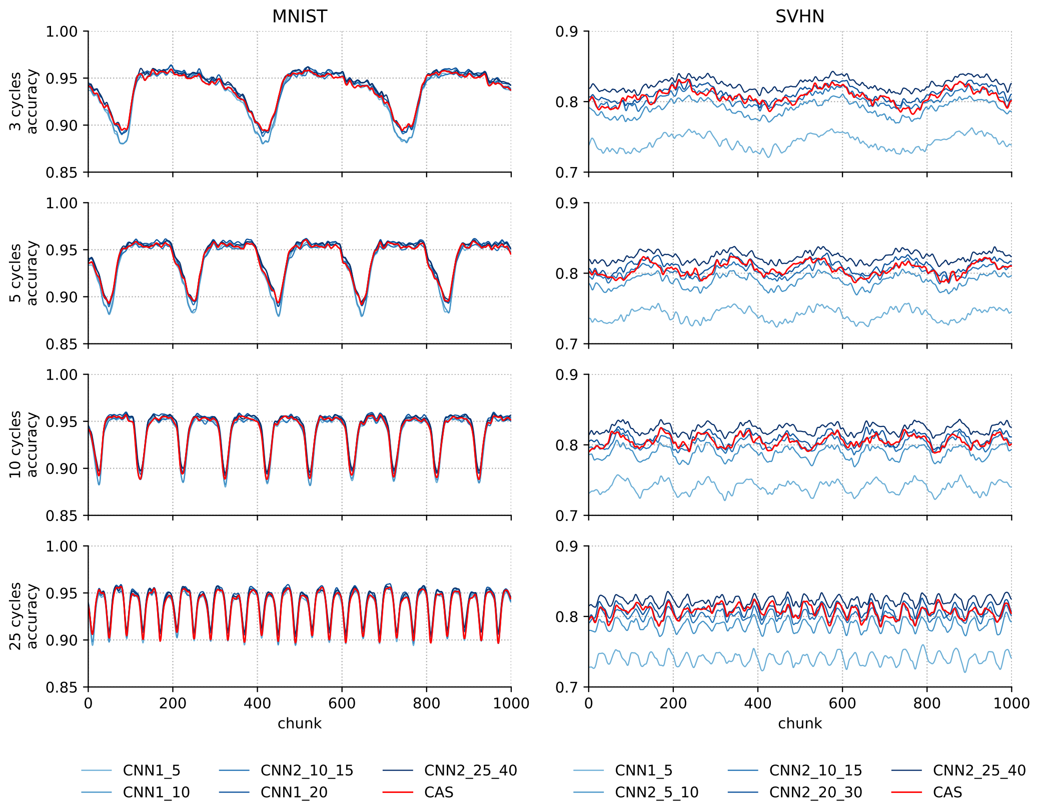

Figure 4.13: *The accuracy of classification with considered architectures (blue) and proposed* CAS *framework (red) for the data streams with the chunk size of 300. The results for* MNIST *are shown on the left and on the right for* SVHN*. The darkest blue indicates the most complex* CNN*, and the lightest blue – the one with lowest time complexity. The rows indicate the number of difficulty changes in the data stream. The results were smoothed using a Gaussian filter.*

complex, and dark color – the most complex architecture. The red color shows the quality obtained by the CAS framework.

In the case of both used datasets, it can be seen that changing the difficulty in the stream affects the quality of the classification. In the case of the MNIST set, all studied architectures achieve similar classification quality, and the difficulty of the recognition task significantly impacts all of them. For streams generated based on SVHN, there is greater variability in the quality achieved by individual architectures, and the changes in accuracy are smoother.

Especially in the case of streams generated from MNIST, it can be seen that the most complex model was used at the moments of the most significant declines in quality. For the second source dataset, the same relationship between stream difficulty and CAS quality is more challenging to observe – however, some adjustments of the selected model dependent on the difficulty are visible.

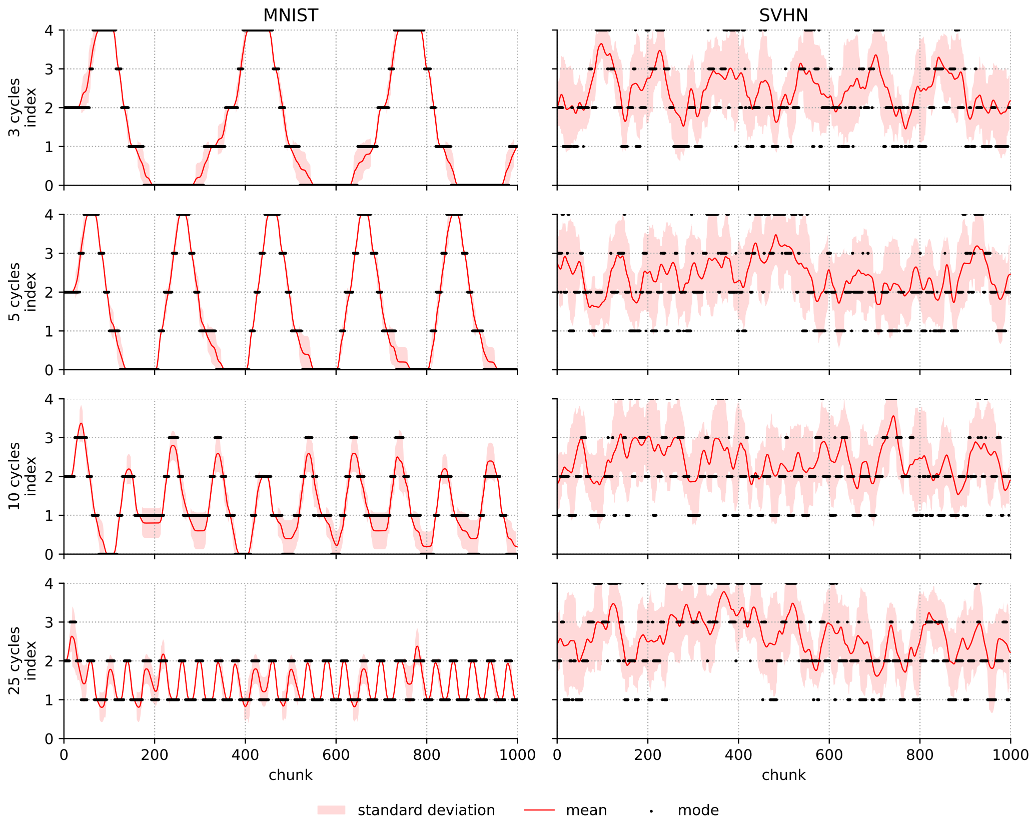

Figure 4.14: *The selected architecture index over stream processing. The results for* MNIST *are shown on the left and for* SVHN *on the right side. The solid red lines indicate the mean index of the selected model, and the bright red area shows the range of one standard deviation from the mean. The black markers show the median value of architecture selected in a given processing point. The rows indicate the number of difficulty changes in the data stream.*

The direct evaluation of the selected architectures is presented in Figure 4.14, showing the indexes of selected architecture at a given processing point, averaged over the repetitions. The value of zero on the vertical axis corresponds to selecting the simplest architecture and the value of four – the most complex one.

In the case of the stream generated based on the MNIST set – especially in the case of the stream with three and five cycles of difficulty change – smooth transitions between subsequent architectures can be seen, which are consistent with the dynamics of changes in the stream. In the case of 10 and 25 cycles, the changes occurred with a high frequency, so that the s hyperparameter, denoting the number of observations out of range before an architecture change occurs, did not allow for a fast enough switching of architectures. As a result, the full spectrum of available architectures was not used. In the case of streams generated based on the SVHN set, smooth transitions strongly correlated with the variability of stream difficulty are no longer observed.

In this experiment, it was possible to observe that changing the difficulty of the processed domain affects the quality of classification. Moreover, the results show that the difficulty of the *domain* can be assessed using the metafeatures employing the *certainty* of the considered neural networks. Therefore, the observations motivate the choice of architecture in the proposed framework based on MPS.

The next step of the experimental evaluation was to estimate the benefits of using the proposed approach in terms of the time and computational complexity of the final solution. Table 4.9 presents results of time complexity for MNIST and SVHN datasets and specific architectures selected for those problems. The processing times were measured for a single instance of a dataset.

Table 4.9: *The results of the metrics expressing the time complexity of the selected neural network architectures. The table presents results for* MNIST *and* SVHN *datasets and all models selected for those problems.*

	MODEL	MACC	LATENCY [MS]
MNIST	`CNN1_5`	83 175	10.821
	`CNN1_10`	166 340	15.230
	`CNN2_10_15`	394 955	25.240
	`CNN1_20`	332 670	25.031
	`CNN2_25_40`	1 987 995	83.646
SVHN	`CNN1_5`	310 741	25.012
	`CNN2_5_10`	429 336	29.783
	`CNN2_10_15`	980 183	49.257
	`CNN2_20_30`	2 708 821	98.771
	`CNN2_25_40`	4 011 611	146.475

It was expected that for all considered architectures within a base dataset (MNIST and SVHN), the measures describing the model operation would be in ascending order. Such an observation would confirm the general assumption that a higher *accuracy* model is more computationally expensive – a more complex architecture should show a longer inference time. Such a phenomenon was not observed in the case of two architectures – `CNN2_10_15` and `CNN1_20` for MNIST in *latency* metric. Such inconsistency may result from the specifics of model operation on a hardware device, as the ordering of architectures was done using the time measurement on the desktop computer. This could be avoided by conducting an architecture selection experiment on the hardware device.

The final results presenting classification quality, processing time, and TTAL measure are presented in Table 4.10 for all generated data streams.

The first columns describe the data stream – a combination of the original dataset, chunk size (CS), and number of cycles of difficulty variation. The following columns show the mean results of *accuracy*, MACC, and *latency* for the proposed CAS approach. The latency in the presented table describes the processing of the entire data chunk. The results include the difference between the metric value for the reference method in parentheses, with the reference indicating the most complex of the considered architectures (`CNN2_25_40` for both base datasets). The last column shows the TTAL measure.

The variability in the TTAL measure of two types of data streams based on two significantly different datasets results mainly from the relative *accuracy* loss for these two problems. In the case of MNIST-based streams, the architecture selection with CAS framework

Table 4.10: *The table showing the average result from a second experiment for the streams generated based on* MNIST *and* SVHN *datasets. The columns describe the metrics for* CAS *with a difference from the reference architecture – the best one in a considered pool – in parenthesis. The final column presents the* TTAL *measure.*

DATA STREAM			ACC	LATENCY [S]	MACC (10^9)	TTAL
MNIST	CS 50	3 cycles	0.938 (-0.002)	1.497 (-2.685)	0.027 (-0.072)	203.507
		5 cycles	0.938 (-0.002)	1.470 (-2.713)	0.026 (-0.073)	189.081
		10 cycles	0.938 (-0.002)	1.241 (-2.941)	0.020 (-0.080)	258.983
		25 cycles	0.936 (-0.003)	1.164 (-3.018)	0.018 (-0.082)	183.636
	CS 150	3 cycles	0.938 (-0.002)	4.481 (-8.066)	0.083 (-0.215)	220.977
		5 cycles	0.941 (-0.002)	3.639 (-8.908)	0.059 (-0.239)	255.219
		10 cycles	0.938 (-0.003)	3.074 (-9.473)	0.042 (-0.256)	234.464
		25 cycles	0.936 (-0.003)	3.120 (-9.427)	0.044 (-0.255)	174.711
	CS 300	3 cycles	0.938 (-0.002)	8.467 (-16.627)	0.154 (-0.443)	200.468
		5 cycles	0.940 (-0.002)	7.963 (-17.131)	0.139 (-0.457)	234.060
		10 cycles	0.940 (-0.003)	5.441 (-19.653)	0.070 (-0.527)	221.728
		25 cycles	0.936 (-0.004)	5.901 (-19.193)	0.080 (-0.516)	163.837
	CS 500	3 cycles	0.936 (-0.002)	15.007 (-26.816)	0.280 (-0.714)	193.112
		5 cycles	0.938 (-0.002)	14.264 (-27.559)	0.258 (-0.736)	199.600
		10 cycles	0.937 (-0.003)	9.567 (-32.256)	0.127 (-0.867)	211.779
		25 cycles	0.934 (-0.004)	9.821 (-32.002)	0.133 (-0.861)	152.490
SVHN	CS 50	3 cycles	0.806 (-0.018)	3.665 (-3.659)	0.089 (-0.111)	12.714
		5 cycles	0.805 (-0.018)	3.635 (-3.688)	0.088 (-0.112)	12.926
		10 cycles	0.802 (-0.020)	3.443 (-3.881)	0.082 (-0.118)	12.944
		25 cycles	0.802 (-0.020)	3.346 (-3.978)	0.079 (-0.121)	13.221
	CS 150	3 cycles	0.801 (-0.023)	8.878 (-13.093)	0.201 (-0.401)	14.316
		5 cycles	0.800 (-0.023)	8.951 (-13.020)	0.204 (-0.398)	14.265
		10 cycles	0.800 (-0.022)	9.251 (-12.720)	0.213 (-0.389)	13.793
		25 cycles	0.801 (-0.022)	9.108 (-12.863)	0.209 (-0.393)	14.232
	CS 300	3 cycles	0.807 (-0.017)	23.360 (-20.582)	0.579 (-0.625)	11.477
		5 cycles	0.806 (-0.018)	23.148 (-20.795)	0.572 (-0.632)	11.518
		10 cycles	0.805 (-0.018)	22.970 (-20.972)	0.566 (-0.637)	11.862
		25 cycles	0.807 (-0.016)	25.706 (-18.236)	0.652 (-0.551)	10.024
	CS 500	3 cycles	0.803 (-0.021)	33.836 (-39.402)	0.804 (-1.201)	12.605
		5 cycles	0.806 (-0.019)	37.800 (-35.438)	0.929 (-1.077)	11.354
		10 cycles	0.807 (-0.017)	40.576 (-32.661)	1.017 (-0.989)	10.701
		25 cycles	0.803 (-0.019)	36.881 (-36.357)	0.900 (-1.106)	11.631

resulted in a drop in *accuracy* of only fractions of a percent, while in the case of SVHN – this difference was about two percent. Meanwhile, the difference in processing time expressed in *latency* and MACC remains similar across those two problems.

The results showed that based on a model's certainty measure and switching architectures in the proposed CAS *framework, it is possible to minimize the processing time by appropriate architecture selection without a meaningful drop in classification quality. This benefit was especially visible for the simpler* MNIST*-based data streams, where the drop in classification accuracy at a level of fractions of a percent was associated with the decrease in estimated latency by almost 200% and in estimated* MACC *by over 280%.*

Chapter 5

Conclusions and future works

This dissertation addresses the employment of metafeature analysis in non-stationary data stream processing, focusing on the prominent tasks of concept drift detection and data stream classification. The conducted research revealed the significance of considering the data distribution non-stationarity when processing data streams and highlighted the potential utility of metafeatures in this research area. As a result of the performed studies, three methods of concept drift detection and three methods for data stream classification were proposed. The methods were evaluated under a wide range of possible conditions impacting the changes observed in a data stream and the difficulty of the recognition task.

The results of the work presented in this dissertation allow to substantiate the research hypothesis:

> *It is possible to propose methods employing metafeature analysis for concept drift detection and classification of the non-stationary data streams that demonstrate significantly better or statistically dependent recognition quality compared to state-of-the-art approaches.*

All proposed approaches showed the recognition quality statistically significantly better or equivalent to identified reference methods or offered additional benefits in data stream processing tasks. The effects of the individual research objectives are discussed below.

1. Proposal of an *implicit concept drift detector* analyzing the time variability of the *classification task complexity measures* calculated for disjoint data chunks.

This objective was achieved by proposing the *Complexity-based Drift Detector* (C2D) method, which uses complexity measures to identify concepts. This method uses an ensemble of one-class classifiers to recognize concepts based on metafeatures calculated for separate chunks in the data stream. The research was performed on synthetic streams, allowing for a comparison of the proposed approach with a wide range of reference drift detectors using three *drift detection error* measures. A proposed method was also utilized for concept drift detection in real-world data streams. Although the qualitative assessment of its performance was not possible, a visualization of the variability of individual problem complexity measures allowed for *explanation* of the detections. Section 3.1 described the proposed approach and the conducted experiments.

The results of the work on the *Complexity-based Drift Detector* were published in the *Neurocomputing* journal [118].

2. Proposal of an *implicit concept drift detector* analyzing *drift magnitude measures* integrated using the *ensemble learning paradigm.*

The motivation for defining this objective was the work by Micevska et al. [163], showing the potential of *concept drift magnitude* measures identifying changes in individual features. The goal was met by the proposition of *Statistical Drift Detection Ensemble* (SDDE), a method using *drift magnitude* and *conditioned marginal covariate drift* metafeatures calculated on problem subspaces. The measures for specific features were integrated using an ensemble mechanism, allowing for effective drift detection. In the extensive research on synthetic data streams, the method showed a high detection quality compared to *state-of-the-art* methods. Section 3.2 described the proposed approach and the conducted experiments.

The results of the work on the *Statistical Drift Detection Ensemble* were published in the *Knowledge-Based Systems* journal [126].

3. Proposal of an *unsupervised drift detection method* analyzing the distribution of *activations* from the last layer of a *deterministic neural network*.

The objective was achieved by the proposition of *Parallel Activations Drift Detector* (PADD). The proposed approach used a series of transformations offered by a deterministic and untrained neural network and used its output as metafeatures, allowing effective detection in an unsupervised setting. This approach was tested on high-dimensional streams, where the prospect for effective use of the neural networks was identified. The proposed approach was compared with supervised and unsupervised methods. The results showed the high quality of concept drift recognition and confirmed that it states a valuable addition to the pool of existing *state-of-the-art* methods. Section 3.3 described the proposed approach and the conducted experiments.

The research results were presented at the *Discovering Drift Phenomena in Evolving Landscape* workshop, held as part of the *International Conference on Knowledge Discovery and Data Mining* (KDD 2024) [122].

4. Proposal of an *ensemble method* for *classification of data streams* analyzing the distributions of *statistical metafeatures* calculated for subsequent data chunks to identify recurring concepts.

This objective was achieved by proposing the *Metafeature Concept Selector* (MCS) method. The method used a set of statistical metafeatures calculated on disjoint chunks of data streams. These metafeatures described a new metaproblem, in which the goal is to assign an identifier to a recurring concept using one-class classifiers based on the metafeatures of a given chunk. The proposed solution was evaluated on a wide pool of streams with a recurring concept. The method has been integrated with three commonly used methods of classifying data streams, showing its ability to increase recognition quality in most studied cases. Section 4.1 described the proposed approach and the conducted experiments.

A review study conducted to select appropriate statistical measures and investigate the potential of metafeatures in the concept identification task has been published in the *Machine Learning* journal [121]. The *Metafeature Concept Selector* method was presented at an *IncrLearn 2024* workshop, held as part of the *IEEE International Conference on Data Mining* (ICDM 2024) [120].

5. Proposal of a *classification* method for compensating the bias of baseline classifiers when processing the data streams with dynamic changes in *imbalance ratio*, using the prior probability estimated based on the *metafeatures* of the processed data chunk.

The objective was achieved by developing *Prior Probability Assisted Classifier* (2PAC) method, which uses *prior probability estimation* to compensate the classifier's original predictions in dynamically imbalanced data streams. The research used three approaches to estimating the prior probability based on metafeatures calculated for previous chunks of data. *Dynamic Statistical Concept Analysis* (DSCA) method was identified as a valuable estimation method, which works particularly well in the case of *discrete dynamically imbalanced data streams*. The research to validate the effectiveness of the 2PAC method used data streams with various characteristics of changes in the imbalance ratio. In all evaluated streaming environments, the proposed 2PAC approach resulted in a statistically significant increase in the average recognition quality compared to the baseline classifier. Section 4.2 described the proposed approach and the conducted experiments.

The results of the work carried out to achieve the above objective were published in the proceedings of the *International Joint Conference on Neural Networks* in 2021 (IJCNN 2021), presenting the *Dynamic Statistical Concept Analysis* [124] method for prior probability estimation, and in 2022 (IJCNN 2022), presenting the *Prior Probability Assisted Classifier* [125].

6. Proposal of a framework for processing data streams with a time-varying level of difficulty, allowing for the selection of an appropriate neural network architecture based on the analysis of the model's *certainty*.

The objective was achieved by the proposal of the *Certainty-based Architecture Selection Framework* (CAS). In the environment of the time-varying level of problem difficulty, the proposed method searches for the appropriate architecture from the pool of pre-trained models, bringing benefits in processing time without a significant decrease in classification quality. The presented solution did not directly intend to improve the classification quality obtained by the model but shifted the focus to a trade-off between the usage of computational resources and the recognition quality. The objective extended the scope of the research, aiming to broaden the scope of metafeature's usage in non-stationary data streams.

The metafeatures used in the approach are defined based on the deterministic output of a pretrained neural networks, minimizing the computational overhead associated with estimating the problem's difficulty. The research used semi-synthetic *computer vision* data streams, where the original images were sampled according to their estimated difficulty. Section 4.3 described the proposed approach and the conducted experiments.

The results of the work on the *Certainty-based Architecture Selection Framework* were published in the *IEEE Access* journal [123].

Future directions

The methods developed in this dissertation can potentially be extended, increasing their reliability and utility. Each of the proposed concept drift detection methods, a necessary element for the proper operation of the method was the selection of hyperparameters indicating its sensitivity to data stream changes. A beneficial direction for future research would be the automatic selection of method configuration, allowing to avoid the need for its tuning for the considered problem.

A similar improvement can be implemented in the MCS method, in which the detection of concept drifts is used indirectly. The operation of the method also depends on the hyperparameters indicating the sensitivity to changes. The MCS method can be further developed with a more complex mechanism of integrating the responses of classifier ensemble members, allowing the method to be used in gradual drifts – when the moment of co-occurrence of objects from two consecutive concepts is visible. The 2PAC method can be extended with a mechanism for identifying the types of prior probability changes in the data stream – so that the best estimation method is dynamically selected. In the CAS method, it is worth considering the automatic adjustment of the switching thresholds of individual models. A valuable extension of the method would be to investigate other functions for assessing the difficulty of the problem that would cause minimal computational overhead.

Finally, the future works, after the thorough examination of the existing vulnerabilities observed in the data streams, can focus on other types of data non-stationarity – such as *anomaly detection*, *novel class discovery*, and *open set recognition.* All those potential future research areas extend the vulnerabilities of processing methods further beyond the canonical concept drift.

Publications

The selected parts of the methods presented in this dissertation have already been published in:

- Komorniczak, J., Zyblewski, P., Ksieniewicz, P. (2021, July). Prior probability estimation in dynamically imbalanced data streams. In 2021 International Joint Conference on Neural Networks (IJCNN) (pp. 1-7). IEEE. (CORE B, MNiSW 140)
- Komorniczak, J., Zyblewski, P., Ksieniewicz, P. (2022, July). Imbalanced data stream classification assisted by prior probability estimation. In 2022 International Joint Conference on Neural Networks (IJCNN) (pp. 1-8). IEEE. (CORE B, MNiSW 140)
- Komorniczak, J., Zyblewski, P., Ksieniewicz, P. (2022). Statistical drift detection ensemble for batch processing of data streams. Knowledge-Based Systems, 252, 109380. (IF 8.8, MNiSW 200)
- Komorniczak, J., Ksieniewicz, P. (2023). Complexity-based drift detection for non-stationary data streams. Neurocomputing, 552, 126554. (IF 5.5, MNiSW 140)
- Komorniczak, J., Ksieniewicz, P. (2024). On metafeatures' ability of implicit concept identification. Machine Learning, 113(10), 7931-7966. (IF 4.3, MNiSW 140)
- Komorniczak, J., Puślecki, T., Ksieniewicz, P., Walkowiak, K. (2024). Certainty-based Neural Network Architecture Selection Framework for TinyML Systems. IEEE Access. (IF 3.4, MNiSW 100).
- Komorniczak, J. P., Ksieniewicz. Unsupervised Concept Drift Detection Based on Parallel Activations of Neural Network. Presented on DELTA Workshop, KDD 2024. (CORE A*, MNiSW 200)
- Komorniczak, J., Ksieniewicz, P. (2024, December). Classification of Recurrent Concepts with Metafeature-based Model Selection. In 2024 IEEE International Conference on Data Mining Workshops (ICDMW) (pp. 213-220). IEEE. (CORE A*, MNiSW 200)

Moreover, for the conducted research, the *problexity* Python programming library was developed, allowing for evaluation of the *complexity metafeatures* of supervised machine learning tasks. The package was published as:

- Komorniczak, J., Ksieniewicz, P. (2023). problexity—An open-source Python library for supervised learning problem complexity assessment. Neurocomputing, 521, 126-136. (IF 5.5, MNiSW 140)

Besides its usage in the research presented in this dissertation, the developed package allowed to investigate the relations between problem complexity and classification accuracy. The effects of such work have been published in:

- Komorniczak, J., Ksieniewicz, P., Woźniak, M. (2022, October). Data complexity and classification accuracy correlation in oversampling algorithms. In Fourth International Workshop on Learning with Imbalanced Domains: Theory and Applications (pp. 175-186). PMLR. (CORE A, MNiSW 140)
- Komorniczak, J., Ksieniewicz, P., Woźniak, M. (2023, June). Analysis of the Possibility to Employ Relationship Between the Problem Complexity and the Classification Quality as Model Optimization Proxy. In International Conference on Computer Recognition Systems (pp. 71-82). Cham: Springer Nature Switzerland. (MNiSW 20)

For reliable method evaluation, the semi-synthetic data stream generation method was developed, which was published in:

- Komorniczak, J., Ksieniewicz, P. (2022). Data stream generation through real concept's interpolation. In ESANN. (CORE B, MNiSW 70)

The generator is now part of a *stream-learn* package [133], which was extended and maintained during the preparation of this dissertation.

During the research considering non-stationary data stream processing additional works in the area of *open set recognition* were conducted. The mentioned field shares similarities with non-stationary data streams in terms of the non-predictability of the open environment. This interest and investigation of possible solutions resulted in the following publications:

- Komorniczak, J., Ksieniewicz, P. (2024). torchosr—A PyTorch extension package for Open Set Recognition models evaluation in Python. Neurocomputing, 566, 127047. (IF 5.5, MNiSW 140)
- Komorniczak, J., Ksieniewicz, P. (2024). Distance profile layer for binary classification and density estimation. Neurocomputing, 579, 127436. (IF 5.5, MNiSW 140)
- Komorniczak, J., Ksieniewicz, P. (2025). Taking class imbalance into account in open set recognition evaluation. Neural Computing and Applications, 1-15. (IF 4.5, MNiSW 100)

During the research for this dissertation, other works regarding text analysis, fake news detection, and data stream processing have been conducted, resulting in the following co-authored publications:

- Komorniczak, J., Wojciechowski, S., Klikowski, J., Kozik, R., Choraś, M. (2023, August). Analysis of Extractive Text Summarization Methods as a Binary Classification Problem. In Computational Intelligence in Security for Information Systems Conference (pp. 91-100). Cham: Springer Nature Switzerland. (MNiSW 20)
- Kozik, R., Komorniczak, J., Ksieniewicz, P., Pawlicka, A., Pawlicki, M., Choraś, M. (2023, August). SWAROG Project Approach to Fake News Detection Problem. In Computational Intelligence in Security for Information Systems Conference (pp. 79-88). Cham: Springer Nature Switzerland. (MNiSW 20)
- Kątek, G., Gackowska, M., Komorniczak, J., Ksieniewicz, P., Kozik, R., Pawlicki, M., Choraś, M. (2024, April). Involving Society to Protect Society from Fake News and Disinformation: Crowdsourced Datasets and Text Reliability Assessment. In Asian Conference on Intelligent Information and Database Systems (pp. 384-395). Singapore: Springer Nature Singapore. (CORE B, MNiSW 70)
- Kozik, R., Kątek, G., Gackowska, M., Kula, S., Komorniczak, J., Ksieniewicz, P., Choraś, M. (2024). Towards explainable fake news detection and automated content credibility assessment: Polish internet and digital media use-case. Neurocomputing, 608, 128450. (IF 5.5, MNiSW 140)
- Wojtachnia, K., Komorniczak, J., Ksieniewicz, P. (2023, June). Incremental Extreme Learning Machine for Binary Data Stream Classification. In International Conference on Computer Recognition Systems (pp. 35-44). Cham: Springer Nature Switzerland. (MNiSW 20)

Finally, at the time of finalizing this dissertation, there are two publications ongoing the review process:

- Komorniczak, J., Zyblewski, P., Ksieniewicz, P., *Structuring the Processing Frameworks for Data Stream Evaluation and Application*, submitted to *Pattern Recognition Journal* in November 2024.
- Komorniczak, J. *Describing Nonstationary Data Streams in Frequency Domain*, submitted to *Knowledge and Information Systems Journal* in February 2025.

Bibliography

[1] Abdulla Amin Aburomman and Mamun Bin Ibne Reaz. A survey of intrusion detection systems based on ensemble and hybrid classifiers. *Computers & security*, 65:135–152, 2017.

[2] Stavros P Adam, Stamatios-Aggelos N Alexandropoulos, Panos M Pardalos, and Michael N Vrahatis. No free lunch theorem: A review. *Approximation and optimization: Algorithms, complexity and applications*, pages 57–82, 2019.

[3] Charu C Aggarwal. *Data streams: models and algorithms*, volume 31. Springer Science & Business Media, 2007.

[4] Supriya Agrahari and Anil Kumar Singh. Concept drift detection in data stream mining: A literature review. *Journal of King Saud University-Computer and Information Sciences*, 34(10):9523–9540, 2022.

[5] Gabriel Aguiar, Bartosz Krawczyk, and Alberto Cano. A survey on learning from imbalanced data streams: taxonomy, challenges, empirical study, and reproducible experimental framework. *Machine learning*, 113(7):4165–4243, 2024.

[6] David W Aha, Dennis Kibler, and Marc K Albert. Instance-based learning algorithms. *Machine learning*, 6:37–66, 1991.

[7] Edesio Alcobaça, Felipe Siqueira, Adriano Rivolli, Luís PF Garcia, Jefferson T Oliva, and André CPLF De Carvalho. Mfe: Towards reproducible meta-feature extraction. *Journal of Machine Learning Research*, 21(111):1–5, 2020.

[8] Shawkat Ali and Kate A Smith. On learning algorithm selection for classification. *Applied Soft Computing*, 6(2):119–138, 2006.

[9] Ethem Alpaydin. *Introduction to machine learning.* MIT press, 2020.

[10] Bartlomiej Antosik and Marek Kurzynski. New measures of classifier competence-heuristics and application to the design of multiple classifier systems. In *Computer Recognition Systems 4*, pages 197–206. Springer, 2011.

[11] Stephen H. Bach and Marcus A. Maloof. Paired learners for concept drift. In *2008 Eighth IEEE International Conference on Data Mining*, pages 23–32, 2008.

[12] Manuel Baena-Garcıa, José del Campo-Ávila, Raul Fidalgo, Albert Bifet, Ricard Gavalda, and Rafael Morales-Bueno. Early drift detection method. In *Fourth international workshop on knowledge discovery from data streams*, volume 6, pages 77–86. Citeseer, 2006.

[13] Ricardo Baeza-Yates. Lecture held at the academia europaea building bridges conference 2022: An introduction to responsible AI. *European Review*, 31(4):406–421, 2023.

[14] Victor H Barella, Luis PF Garcia, Marcilio CP de Souto, Ana C Lorena, and André CPLF de Carvalho. Assessing the data complexity of imbalanced datasets. *Information Sciences*, 553:83–109, 2021.

[15] Victor H Barella, Luís PF Garcia, Marcilio P de Souto, Ana C Lorena, and André de Carvalho. Data complexity measures for imbalanced classification tasks. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2018.

[16] Dipto Barman, Ziyi Guo, and Owen Conlan. The dark side of language models: Exploring the potential of llms in multimedia disinformation generation and dissemination. *Machine Learning with Applications*, page 100545, 2024.

[17] Roberto Barros and Silas Santos. A large-scale comparison of concept drift detectors. *Information Sciences*, 451-452, 07 2018.

[18] Mohammad Mahdi Bejani and Mehdi Ghatee. A systematic review on overfitting control in shallow and deep neural networks. *Artificial Intelligence Review*, 54(8):6391–6438, 2021.

[19] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[20] Rudolf Beran. Minimum hellinger distance estimates for parametric models. *The annals of Statistics*, pages 445–463, 1977.

[21] Vasudha Bhatnagar, Manju Bhardwaj, Shivam Sharma, and Sufyan Haroon. Accuracy–diversity based pruning of classifier ensembles. *Progress in Artificial Intelligence*, 2(2):97–111, 2014.

[22] Albert Bifet. Classifier concept drift detection and the illusion of progress. In *Artificial Intelligence and Soft Computing: 16th International Conference, ICAISC 2017, Zakopane, Poland, June 11-15, 2017, Proceedings, Part II 16*, pages 715–725. Springer, 2017.

[23] Albert Bifet, Gianmarco de Francisci Morales, Jesse Read, Geoff Holmes, and Bernhard Pfahringer. Efficient online evaluation of big data stream classifiers. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 59–68, 2015.

[24] Albert Bifet and Ricard Gavalda. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM international conference on data mining*, pages 443–448. SIAM, 2007.

[25] Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. Leveraging bagging for evolving data streams. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 135–150. Springer, 2010.

[26] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavalda. New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 139–148, 2009.

[27] Christopher M Bishop. *Neural networks for pattern recognition.* Oxford university press, 1995.

[28] Frederic Branchaud-Charron, Andrew Achkar, and Pierre-Marc Jodoin. Spectral metric for dataset complexity assessment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3215–3224, 2019.

[29] James H Bray and Scott E Maxwell. *Multivariate analysis of variance.* Number 54. Sage, 1985.

[30] Pavel Brazdil, Christophe Giraud Carrier, Carlos Soares, and Ricardo Vilalta. *Metalearning: Applications to data mining.* Springer Science & Business Media, 2008.

[31] Leo Breiman. Bagging predictors. *Machine learning*, 24:123–140, 1996.

[32] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[33] Alceu S Britto Jr, Robert Sabourin, and Luiz ES Oliveira. Dynamic selection of classifiers—a comprehensive review. *Pattern recognition*, 47(11):3665–3680, 2014.

[34] Rasmus Bro and Age K Smilde. Principal component analysis. *Analytical methods*, 6(9):2812–2831, 2014.

[35] Dariusz Brzezinski and Jerzy Stefanowski. Reacting to different types of concept drift: The accuracy updated ensemble algorithm. *IEEE transactions on neural networks and learning systems*, 25(1):81–94, 2013.

[36] Dariusz Brzezinski, Jerzy Stefanowski, Robert Susmaga, and Izabela Szczech. On the dynamics of classification measures for imbalanced and streaming data. *IEEE transactions on neural networks and learning systems*, 31(8):2868–2878, 2019.

[37] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with GPT-4. *arXiv preprint arXiv:2303.12712*, 2023.

[38] Alberto Cano and Bartosz Krawczyk. Kappa updated ensemble for drifting data stream mining. *Machine Learning*, 109(1):175–218, 2020.

[39] Alberto Cano and Bartosz Krawczyk. ROSE: robust online self-adjusting ensemble for continual learning on imbalanced drifting data streams. *Machine Learning*, pages 1–39, 2022.

[40] Rodolfo C Cavalcante, Leandro L Minku, and Adriano LI Oliveira. Fedd: Feature extraction for explicit concept drift detection in time series. In *2016 International joint conference on neural networks (IJCNN)*, pages 740–747. IEEE, 2016.

[41] Tsan-Ming Choi, Stein W Wallace, and Yulan Wang. Big data analytics in operations management. *Production and operations management*, 27(10):1868–1883, 2018.

[42] Fang Chu and Carlo Zaniolo. Fast and light boosting for adaptive mining of data streams. In *Advances in Knowledge Discovery and Data Mining: 8th Pacific-Asia Conference, PAKDD 2004, Sydney, Australia, May 26-28, 2004. Proceedings 8*, pages 282–292. Springer, 2004.

[43] Corinna Cortes and Vladimir Vapnik. Support vector machine. *Machine learning*, 20(3):273–297, 1995.

[44] Rafael MO Cruz, Robert Sabourin, and George DC Cavalcanti. Dynamic classifier selection: Recent advances and perspectives. *Information Fusion*, 41:195–216, 2018.

[45] Tiago Cunha, Carlos Soares, and André CPLF de Carvalho. Metalearning and recommender systems: A literature review and empirical study on the algorithm selection problem for collaborative filtering. *Information Sciences*, 423:128–144, 2018.

[46] Belur V Dasarathy and Belur V Sheela. A composite classifier system design: Concepts and methodology. *Proceedings of the IEEE*, 67(5):708–713, 1979.

[47] Peter Dayan, Maneesh Sahani, and Grégoire Deback. Unsupervised learning. *The MIT encyclopedia of the cognitive sciences*, pages 857–859, 1999.

[48] Roberto Souto Maior de Barros and Silas Garrido T de Carvalho Santos. An overview and comprehensive comparison of ensembles for concept drift. *Information Fusion*, 52:213–244, 2019.

[49] Emanuele Della Valle, Giacomo Ziffer, Alessio Bernardo, Vitor Cerqueira, and Albert Bifet. Towards time-evolving analytics: Online learning for time-dependent evolving data streams. *Data Science*, 16, 2022.

[50] Jaka Demšar and Zoran Bosnić. Detecting concept drift in data streams using model explanation. *Expert Systems with Applications*, 92:546–559, 2018.

[51] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE signal processing magazine*, 29(6):141–142, 2012.

[52] Aniket Didolkar, Anirudh Goyal, Nan Rosemary Ke, Siyuan Guo, Michal Valko, Timothy Lillicrap, Danilo Rezende, Yoshua Bengio, Michael Mozer, and Sanjeev Arora. Metacognitive capabilities of llms: An exploration in mathematical problem solving. *arXiv preprint arXiv:2405.12205*, 2024.

[53] Pedro Domingos. *The master algorithm: How the quest for the ultimate learning machine will remake our world.* Basic Books, 2015.

[54] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80, 2000.

[55] Pedro Domingos and Geoff Hulten. A general framework for mining massive data streams. *Journal of Computational and Graphical Statistics*, 12(4):945–949, 2003.

[56] Lei Du, Qinbao Song, Lei Zhu, and Xiaoyan Zhu. A selective detector ensemble for concept drift detection. *The Computer Journal*, 58(3):457–471, 06 2014.

[57] Abdussalam Elhanashi, Pierpaolo Dini, Sergio Saponara, and Qinghe Zheng. Advancements in tinyml: Applications, limitations, and impact on iot devices. *Electronics*, 13(17):3562, 2024.

[58] Ryan Elwell and Robi Polikar. Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks*, 22(10):1517–1531, 2011.

[59] EA Feigenbaum, J Lederber, and R Buchanan. Heuristic dendral. In *Proceedings of the Hawaii International Conference on System Sciences*, 1968.

[60] Daniel Gomes Ferrari and Leandro Nunes De Castro. Clustering algorithm selection by meta-learning systems: A new distance-based problem characterization and ranking combination methods. *Information Sciences*, 301:181–194, 2015.

[61] Xavier Ferrer, Tom Van Nuenen, Jose M Such, Mark Coté, and Natalia Criado. Bias and discrimination in ai: a cross-disciplinary perspective. *IEEE Technology and Society Magazine*, 40(2):72–80, 2021.

[62] Sara Fotouhi, Shahrokh Asadi, and Michael W Kattan. A comprehensive data level analysis for cancer diagnosis on imbalanced data. *Journal of biomedical informatics*, 90:103089, 2019.

[63] Teodor Fredriksson, David Issa Mattos, Jan Bosch, and Helena Holmström Olsson. Data labeling: An empirical investigation into industrial challenges and mitigation strategies. In *International Conference on Product-Focused Software Process Improvement*, pages 202–216. Springer, 2020.

[64] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[65] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[66] Isvani Frias-Blanco, José del Campo-Ávila, Gonzalo Ramos-Jimenez, Rafael Morales-Bueno, Agustin Ortiz-Diaz, and Yailé Caballero-Mota. Online and non-parametric drift detection methods based on hoeffding's bounds. *IEEE Transactions on Knowledge and Data Engineering*, 27(3):810–823, 2014.

[67] Richard M Friedberg. A learning machine: Part I. *IBM Journal of Research and Development*, 2(1):2–13, 1958.

[68] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[69] Mohamed Medhat Gaber. Advances in data stream mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):79–85, 2012.

[70] Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In *Brazilian symposium on artificial intelligence*, pages 286–295. Springer, 2004.

[71] Joao Gama, Pedro Pereira Rodrigues, Eduardo Spinosa, and Andre Carvalho. Knowledge discovery from data streams. In *Web Intelligence and Security*, pages 125–138. IOS Press, 2010.

[72] Joao Gama, Raquel Sebastiao, and Pedro Pereira Rodrigues. Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 329–338, 2009.

[73] Joao Gama, Raquel Sebastiao, and Pedro Pereira Rodrigues. On evaluating stream learning algorithms. *Machine learning*, 90:317–346, 2013.

[74] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):1–37, 2014.

[75] Jing Gao, Wei Fan, Jiawei Han, and Philip S Yu. A general framework for mining concept-drifting data streams with skewed distributions. In *Proceedings of the 2007 siam international conference on data mining*, pages 3–14. SIAM, 2007.

[76] Ricard Gavaldà. Machine learning for clinical management: From the lab to the hospital. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4736–4736, 2024.

[77] A Shaji George and AS Hovan George. A review of ChatGPT AI's impact on several business sectors. *Partners universal international innovation journal*, 1(1):9–23, 2023.

[78] Sofie Goethals, David Martens, and Theodoros Evgeniou. The non-linear nature of the cost of comprehensibility. *Journal of Big Data*, 9(1):1–23, 2022.

[79] Heitor M Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrício Enembreck, Bernhard Pfharinger, Geoff Holmes, and Talel Abdessalem. Adaptive random forests for evolving data stream classification. *Machine Learning*, 106:1469–1495, 2017.

[80] Heitor Murilo Gomes, Jean Paul Barddal, Fabrício Enembreck, and Albert Bifet. A survey on ensemble learning for data stream classification. *ACM Computing Surveys (CSUR)*, 50(2):1–36, 2017.

[81] Heitor Murilo Gomes, Jesse Read, and Albert Bifet. Streaming random patches for evolving data stream classification. In *2019 IEEE international conference on data mining (ICDM)*, pages 240–249. IEEE, 2019.

[82] Rafael C Gonzales and Paul Wintz. *Digital image processing.* Addison-Wesley Longman Publishing Co., Inc., 1987.

[83] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[84] Ömer Gözüaçık, Alican Büyükçakır, Hamed Bonab, and Fazli Can. Unsupervised concept drift detection with a discriminative classifier. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 2365–2368, 2019.

[85] Ömer Gözüaçık and Fazli Can. Concept learning using one-class classifiers for implicit drift detection in evolving data streams. *Artificial Intelligence Review*, 54(5):3725–3747, 2021.

[86] Maciej Grzenda, Heitor Murilo Gomes, and Albert Bifet. Delayed labelling evaluation for data streams. *Data Mining and Knowledge Discovery*, 34(5):1237–1266, 2020.

[87] Yufeng Guo, Peng Zhou, Yanping Zhang, and Xin Jiang. Meta-feature-based concept evolution detection on feature streams. In *2023 8th International Conference on Intelligent Computing and Signal Processing (ICSP)*, pages 1995–1998. IEEE, 2023.

[88] Isabelle Guyon, Kristin Bennett, Gavin Cawley, Hugo Jair Escalante, Sergio Escalera, Tin Kam Ho, Núria Macià, Bisakha Ray, Mehreen Saeed, Alexander Statnikov, et al. Design of the 2015 ChaLearn AutoML challenge. In *2015 International joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2015.

[89] Ben Halstead, Yun Sing Koh, Patricia Riddle, Mykola Pechenizkiy, and Albert Bifet. Combining diverse meta-features to accurately identify recurring concept drift in data streams. *ACM Transactions on Knowledge Discovery from Data*, 17(8):1–36, 2023.

[90] Yoshihiko Hamamoto, Shunji Uchimura, and Shingo Tomita. On the behavior of artificial neural network classifiers in high-dimensional spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(5):571–574, 1996.

[91] Peter Hart. The condensed nearest neighbor rule (corresp.). *IEEE transactions on information theory*, 14(3):515–516, 1968.

[92] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[93] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[94] Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *Knowledge-based systems*, 212:106622, 2021.

[95] Lennart Heim, Andreas Biri, Zhongnan Qu, and Lothar Thiele. Measuring what really matters: Optimizing neural networks for TinyML. *arXiv preprint arXiv:2104.10645*, 2021.

[96] Pascal Hitzler and Md Kamruzzaman Sarker. Neuro-symbolic artificial intelligence: The state of the art. 2022.

[97] Tin Kam Ho and Mitra Basu. Complexity measures of supervised classification problems. *IEEE transactions on pattern analysis and machine intelligence*, 24(3):289–300, 2002.

[98] Marcel Hofmann. Developing a streaming-based architecture for demand prediction of taxi trips in the presence of concept drift. 2019.

[99] Simon Hornblower. Creation and development of democratic institutions in ancient Greece. *Democracy: the unfinished journey*, 508:1–16, 1992.

[100] Hanqing Hu, Mehmed Kantardzic, and Tegjyot S Sethi. No free lunch theorem for concept drift detection in streaming data classification: A review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 10(2):e1327, 2020.

[101] Lianyu Hu, Mudi Jiang, Junjie Dong, Xinying Liu, and Zengyou He. Interpretable Clustering: A Survey. *arXiv preprint arXiv:2409.00743*, 2024.

[102] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2:193–218, 1985.

[103] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 97–106, 2001.

[104] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[105] Nathalie Japkowicz. Assessment metrics for imbalanced learning. *Imbalanced learning: Foundations, algorithms, and applications*, pages 187–206, 2013.

[106] Nan Jiang and Le Gruenwald. Research issues in data stream association rule mining. *ACM Sigmod Record*, 35(1):14–19, 2006.

[107] Yuchen Jiang, Xiang Li, Hao Luo, Shen Yin, and Okyay Kaynak. Quo vadis artificial intelligence? *Discover Artificial Intelligence*, 2(1):4, 2022.

[108] Shubhra Kanti Karmaker, Md Mahadi Hassan, Micah J Smith, Lei Xu, Chengxiang Zhai, and Kalyan Veeramachaneni. AutoML to date and beyond: Challenges and opportunities. *ACM Computing Surveys (CSUR)*, 54(8):1–36, 2021.

[109] Henry Kautz. The third AI summer: AAAI Robert S. Engelmore memorial lecture. *Ai magazine*, 43(1):105–125, 2022.

[110] Mark G Kelly, David J Hand, and Niall M Adams. The impact of changing populations on classifier performance. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 367–371, 1999.

[111] Samina Khalid, Tehmina Khalil, and Shamila Nasreen. A survey of feature selection and feature extraction techniques in machine learning. In *2014 science and information conference*, pages 372–378. IEEE, 2014.

[112] Matloob Khushi, Kamran Shaukat, Talha Mahboob Alam, Ibrahim A Hameed, Shahadat Uddin, Suhuai Luo, Xiaoyan Yang, and Maranatha Consuelo Reyes. A comparative performance analysis of data resampling methods on imbalance medical data. *IEEE Access*, 9:109960–109975, 2021.

[113] Jakub Klikowski. Concept drift detector based on centroid distance analysis. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2022.

[114] Ralf Klinkenberg and Ingrid Renz. Adaptive information filtering: Learning in the presence of concept drifts. *Learning for text categorization*, pages 33–40, 1998.

[115] Aleksandra Knapińska, Piotr Lechowicz, Weronika Węgier, and Krzysztof Walkowiak. Long-term prediction of multiple types of time-varying network traffic using chunk-based ensemble learning. *Applied Soft Computing*, 130:109694, 2022.

[116] J Zico Kolter and Marcus A Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *The Journal of Machine Learning Research*, 8:2755–2790, 2007.

[117] Joanna Komorniczak and Paweł Ksieniewicz. Data stream generation through real concept's interpolation. In *ESANN*, 2022.

[118] Joanna Komorniczak and Paweł Ksieniewicz. Complexity-based drift detection for nonstationary data streams. *Neurocomputing*, 552:126554, 2023.

[119] Joanna Komorniczak and Paweł Ksieniewicz. problexity – an open-source python library for supervised learning problem complexity assessment. *Neurocomputing*, 521:126–136, 2023.

[120] Joanna Komorniczak and Paweł Ksieniewicz. Classification of recurrent concepts with metafeature-based model selection. In *2024 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 213–220. IEEE, 2024.

[121] Joanna Komorniczak and Paweł Ksieniewicz. On metafeatures' ability of implicit concept identification. *Machine Learning*, 113(10):7931–7966, 2024.

[122] Joanna Komorniczak and Paweł Ksieniewicz. Unsupervised concept drift detection based on parallel activations of neural network. In Marco Piangerelli, Bardh Prenkaj, Ylenia Rotalinti, Ananya Joshi, and Giovanni Stilo, editors, *Discovering Drift Phenomena in Evolving Landscapes*, pages 111–127, Cham, 2025. Springer Nature Switzerland.

[123] Joanna Komorniczak, Tobiasz Puślecki, Paweł Ksieniewicz, and Krzysztof Walkowiak. Certainty-based neural network architecture selection framework for TinyML systems. *IEEE Access*, 2024.

[124] Joanna Komorniczak, Paweł Zyblewski, and Paweł Ksieniewicz. Prior probability estimation in dynamically imbalanced data streams. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2021.

[125] Joanna Komorniczak, Paweł Zyblewski, and Paweł Ksieniewicz. Imbalanced data stream classification assisted by prior probability estimation. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2022.

[126] Joanna Komorniczak, Paweł Zyblewski, and Paweł Ksieniewicz. Statistical drift detection ensemble for batch processing of data streams. *Knowledge-Based Systems*, 252:109380, 2022.

[127] Sotiris B Kotsiantis, Ioannis D Zaharakis, and Panayiotis E Pintelas. Machine learning: a review of classification and combining techniques. *Artificial Intelligence Review*, 26:159–190, 2006.

[128] Bartosz Krawczyk, Leandro L Minku, João Gama, Jerzy Stefanowski, and Michał Woźniak. Ensemble learning for data stream analysis: A survey. *Information Fusion*, 37:132–156, 2017.

[129] Bartosz Krawczyk and Michał Woźniak. Weighted naive bayes classifier with forgetting for drifting data streams. In *2015 IEEE International conference on systems, man, and cybernetics*, pages 2147–2152. IEEE, 2015.

[130] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[131] Urszula Krzeszewska, Aneta Poniszewska-Marańda, and Joanna Ochelska-Mierzejewska. Systematic comparison of vectorization methods in classification context. *Applied Sciences*, 12(10):5119, 2022.

[132] Paweł Ksieniewicz. Processing data stream with chunk-similarity model selection. *Applied Intelligence*, 53(7):7931–7956, 2023.

[133] Paweł Ksieniewicz and Paweł Zyblewski. Stream-learn—open-source python library for difficult data stream batch analysis. *Neurocomputing*, 478:11–21, 2022.

[134] Ludmila I Kuncheva. Classifier ensembles for changing environments. In *Multiple Classifier Systems: 5th International Workshop, MCS 2004, Cagliari, Italy, June 9-11, 2004. Proceedings 5*, pages 1–15. Springer, 2004.

[135] Ludmila I Kuncheva. Using control charts for detecting concept change in streaming data. *Bangor University*, page 48, 2009.

[136] Ludmila I Kuncheva. *Combining pattern classifiers: methods and algorithms*. John Wiley & Sons, 2014.

[137] Ludmila I Kuncheva, James C Bezdek, and Robert PW Duin. Decision templates for multiple classifier fusion: an experimental comparison. *Pattern recognition*, 34(2):299–314, 2001.

[138] Ludmila I Kuncheva and Christopher J Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine learning*, 51:181–207, 2003.

[139] Jorma Laaksonen and Erkki Oja. Classification with learning k-nearest neighbors. In *Proceedings of international conference on neural networks (ICNN'96)*, volume 3, pages 1480–1483. IEEE, 1996.

[140] Pat Langley and Stephanie Sage. Induction of selective bayesian classifiers. In *Uncertainty in Artificial Intelligence*, pages 399–406. Elsevier, 1994.

[141] Derrick N Lawley. A generalization of Fisher's Z test. *Biometrika*, 30(1/2):180–187, 1938.

[142] Jeonghoon Lee and Frederic Magoules. Detection of concept drift for learning from stream data. In *2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*, pages 241–245. IEEE, 2012.

[143] Loo Hay Lee, Ek Peng Chew, Suyan Teng, and David Goldsman. Finding the non-dominated pareto set for multi-objective simulation models. *IIE Transactions*, 42(9):656–674, 2010.

[144] Christiane Lemke, Marcin Budka, and Bogdan Gabrys. Metalearning: a survey of trends and technologies. *Artificial intelligence review*, 44:117–130, 2015.

[145] David A Levin and Yuval Peres. *Markov chains and mixing times*, volume 107. American Mathematical Soc., 2017.

[146] Bin Li, Yi-jie Wang, Dong-sheng Yang, Yong-mou Li, and Xing-kong Ma. Faad: an unsupervised fast and accurate anomaly detection method for a multi-dimensional sequence over data stream. *Frontiers of Information Technology & Electronic Engineering*, 20(3):388–404, 2019.

[147] Guang Li, Ren Togo, Takahiro Ogawa, and Miki Haseyama. Dataset complexity assessment based on cumulative maximum scaled area under laplacian spectrum. *Multimedia Tools and Applications*, 81(22):32287–32303, 2022.

[148] Xi Lin, Zhiyuan Yang, Xiaoyuan Zhang, and Qingfu Zhang. Pareto set learning for expensive multi-objective optimization. *Advances in neural information processing systems*, 35:19231–19247, 2022.

[149] Guido Lindner and Rudi Studer. Ast: Support for algorithm selection with a cbr approach. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 418–423. Springer, 1999.

[150] Patrick Lindstrom, Brian Mac Namee, and Sarah Jane Delany. Drift detection using uncertainty distribution divergence. *Evolving Systems*, 4:13–25, 2013.

[151] Anjin Liu, Jie Lu, Feng Liu, and Guangquan Zhang. Accumulating regional density dissimilarity for concept drift detection in data streams. *Pattern Recognition*, 76:256–272, 2018.

[152] Sanmin Liu, Shan Xue, Jia Wu, Chuan Zhou, Jian Yang, Zhao Li, and Jie Cao. Online active learning for drifting data streams. *IEEE Transactions on Neural Networks and Learning Systems*, 34(1):186–200, 2021.

[153] Jesus L Lobo, Javier Del Ser, Ibai Laña, Miren Nekane Bilbao, and Nikola Kasabov. Drift detection over non-stationary data streams using evolving spiking neural networks. In *Intelligent Distributed Computing XII*, pages 82–94. Springer, 2018.

[154] Ana C Lorena, Luís PF Garcia, Jens Lehmann, Marcilio CP Souto, and Tin Kam Ho. How complex is your classification problem? A survey on measuring classification complexity. *ACM Computing Surveys (CSUR)*, 52(5):1–34, 2019.

[155] Ana C Lorena, Aron I Maciel, Péricles BC de Miranda, Ivan G Costa, and Ricardo BC Prudêncio. Data complexity meta-features for regression problems. *Machine Learning*, 107:209–246, 2018.

[156] Gilles Louppe and Pierre Geurts. Ensembles on random patches. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2012, Bristol, UK, September 24-28, 2012. Proceedings, Part I 23*, pages 346–361. Springer, 2012.

[157] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[158] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE transactions on knowledge and data engineering*, 31(12):2346–2363, 2018.

[159] Bruno Maciel, Silas Santos, and Roberto Barros. A lightweight concept drift detection ensemble. 11 2015.

[160] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pages 346–357. Elsevier, 2002.

[161] Dragos D Margineantu and Thomas G Dietterich. Pruning adaptive boosting. In *ICML*, volume 97, pages 211–218. Citeseer, 1997.

[162] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[163] Simona Micevska, Ahmed Awad, and Sherif Sakr. Sddm: an interpretable statistical concept drift detection method for data streams. *Journal of Intelligent Information Systems*, 56, 06 2021.

[164] Ryszard Stanislaw Michalski, Jaime Guillermo Carbonell, and Tom M Mitchell. *Machine learning: An artificial intelligence approach.* Springer Science & Business Media, 2013.

[165] Michael T Mills and Nikolaos G Bourbakis. Graph-based methods for natural language processing and understanding—a survey and analysis. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 44(1):59–71, 2013.

[166] Leandro L Minku, Allan P White, and Xin Yao. The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Transactions on knowledge and Data Engineering*, 22(5):730–742, 2009.

[167] Leandro L Minku and Xin Yao. Ddd: A new ensemble approach for dealing with concept drift. *IEEE transactions on knowledge and data engineering*, 24(4):619–633, 2011.

[168] Marvin Minsky and Seymour A. Papert. *Perceptrons: An Introduction to Computational Geometry.* The MIT Press, 1969.

[169] Tom Michael Mitchell. *Version spaces: an approach to concept learning.* Stanford University, 1979.

[170] Jacob Montiel, Jesse Read, Albert Bifet, and Talel Abdessalem. Scikit-multiflow: A multi-output streaming framework. *Journal of Machine Learning Research*, 19(72):1–5, 2018.

[171] Mario A Muñoz, Laura Villanova, Davaatseren Baatar, and Kate Smith-Miles. Instance spaces for machine learning classification. *Machine Learning*, 107:109–147, 2018.

[172] Shanmugavelayutham Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2):117–236, 2005.

[173] Feng Nan, Joseph Wang, and Venkatesh Saligrama. Pruning random forests for prediction on a budget. *Advances in neural information processing systems*, 29, 2016.

[174] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 4. Granada, 2011.

[175] Hai-Long Nguyen, Yew-Kwong Woon, and Wee-Keong Ng. A survey on data stream clustering and classification. *Knowledge and information systems*, 45:535–569, 2015.

[176] Kyosuke Nishida and Koichiro Yamauchi. Adaptive classifiers-ensemble system for tracking concept drift. In *2007 International Conference on Machine Learning and Cybernetics*, volume 6, pages 3607–3612. IEEE, 2007.

[177] Aytuğ Onan, Serdar Korukoğlu, and Hasan Bulut. A hybrid ensemble pruning approach based on consensus clustering and multi-objective evolutionary algorithm for sentiment classification. *Information Processing & Management*, 53(4):814–833, 2017.

[178] Nikunj C Oza and Stuart J Russell. Online bagging and boosting. In *International workshop on artificial intelligence and statistics*, pages 229–236. PMLR, 2001.

[179] Nikunj C Oza and Kagan Tumer. Classifier ensembles: Select real-world applications. *Information fusion*, 9(1):4–20, 2008.

[180] Ewan S Page. Continuous inspection schemes. *Biometrika*, 41(1/2):100–115, 1954.

[181] Emanuele Panigati, F. Schreiber, and C. Zaniolo. Data streams and data stream management systems and languages. pages 93–111, 2015.

[182] Shraddha Parab and Supriya Bhalerao. Choosing statistical test. *International journal of Ayurveda research*, 1(3):187, 2010.

[183] Megha Ashok Patil, Sunil Kumar, Sandeep Kumar, and Muskan Garg. Concept drift detection for social media: A survey. In *2021 3rd international conference on advances in computing, communication control and networking (ICAC3N)*, pages 12–16. IEEE, 2021.

[184] KC Sreedharan Pillai. Some new test criteria in multivariate analysis. *The Annals of Mathematical Statistics*, pages 117–121, 1955.

[185] Robi Polikar, Lalita Upda, Satish S Upda, and Vasant Honavar. Learn++: An incremental learning algorithm for supervised neural networks. *IEEE transactions on systems, man, and cybernetics, part C (applications and reviews)*, 31(4):497–508, 2001.

[186] Andreas L Prodromidis and Salvatore J Stolfo. Cost complexity-based pruning of ensemble classifiers. *Knowledge and Information Systems*, 3:449–469, 2001.

[187] J. Ross Quinlan. Learning decision tree classifiers. *ACM Computing Surveys (CSUR)*, 28(1):71–72, 1996.

[188] J Ross Quinlan. *C4.5: programs for machine learning.* Elsevier, 2014.

[189] Sergio Ramírez-Gallego, Bartosz Krawczyk, Salvador García, Michał Woźniak, and Francisco Herrera. A survey on data preprocessing for data stream mining: Current status and future directions. *Neurocomputing*, 239:39–57, 2017.

[190] Matthias Reif, Faisal Shafait, and Andreas Dengel. Meta2-features: Providing meta-learners more information. In *35th German Conference on Artificial Intelligence.* Citeseer, 2012.

[191] John R Rice. The algorithm selection problem. In *Advances in computers*, volume 15, pages 65–118. Elsevier, 1976.

[192] Adriano Rivolli, Luís PF Garcia, Carlos Soares, Joaquin Vanschoren, and André CPLF de Carvalho. Meta-features for meta-learning. *Knowledge-Based Systems*, 240:108101, 2022.

[193] Lior Rokach. Ensemble-based classifiers. *Artificial intelligence review*, 33:1–39, 2010.

[194] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[195] Neelam Rout, Debahuti Mishra, and Manas Kumar Mallick. Handling imbalanced data: a survey. In *International Proceedings on Advances in Soft Computing, Intelligent Systems and Applications: ASISA 2016*, pages 431–443. Springer, 2018.

[196] Samarendra Nath Roy. On a heuristic method of test construction and its use in multivariate analysis. *The Annals of Mathematical Statistics*, 24(2):220–238, 1953.

[197] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[198] David E Rumelhart, James L McClelland, PDP Research Group, et al. Parallel distributed processing, 1986.

[199] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

[200] Seref Sagiroglu and Duygu Sinanc. Big data: A review. In *2013 international conference on collaboration technologies and systems (CTS)*, pages 42–47. IEEE, 2013.

[201] Walter J Scheirer, Anderson de Rezende Rocha, Archana Sapkota, and Terrance E Boult. Toward open set recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(7):1757–1772, 2012.

[202] Tegjyot Singh Sethi and Mehmed Kantardzic. Don't pay for validation: Detecting drifts from unlabeled data using margin density. *Procedia Computer Science*, 53:103–112, 2015.

[203] Tegjyot Singh Sethi, Mehmed Kantardzic, and Hanquing Hu. A grid density based framework for classifying streaming data in the presence of concept drift. *Journal of Intelligent Information Systems*, 46:179–211, 2016.

[204] Burr Settles. Active learning literature survey. *University of Wisconsin-Madison Department of Computer Sciences*, 2009.

[205] Lloyd Shapley and Bernard Grofman. Optimizing group judgmental accuracy in the presence of interdependencies. *Public Choice*, 43(3):329–343, 1984.

[206] Yeshayahu Shen and David Gil. How language influences the way we categorize hybrids. In *Handbook of categorization in cognitive science*, pages 1177–1200. Elsevier, 2017.

[207] Amanpreet Singh, Narina Thakur, and Aakanksha Sharma. A review of supervised machine learning algorithms. In *2016 3rd international conference on computing for sustainable global development (INDIACom)*, pages 1310–1315. Ieee, 2016.

[208] Kate A Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)*, 41(1):1–25, 2009.

[209] Symone G Soares and Rui Araújo. A dynamic and on-line ensemble regression for changing environments. *Expert Systems with Applications*, 42(6):2935–2948, 2015.

[210] Piotr Sobolewski and Michał Woźniak. Concept drift detection and model selection with simulated recurrence and ensembles of statistical detectors. *J. Univers. Comput. Sci.*, 19(4):462–483, 2013.

[211] Xiuyao Song, Mingxi Wu, Christopher Jermaine, and Sanjay Ranka. Statistical change detection for multi-dimensional data. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 667–676, 2007.

[212] Vinicius MA Souza, Denis M dos Reis, Andre G Maletzke, and Gustavo EAPA Batista. Challenges in benchmarking stream learning algorithms with real-world data. *Data Mining and Knowledge Discovery*, 34(6):1805–1858, 2020.

[213] Eduardo J Spinosa, André Ponce de Leon F. de Carvalho, and Joao Gama. Olindda: A cluster-based approach for detecting novelty and concept drift in data streams. In *Proceedings of the 2007 ACM symposium on Applied computing*, pages 448–452, 2007.

[214] Katarzyna Stapor, Paweł Ksieniewicz, Salvador García, and Michał Woźniak. How to design the fair experimental classifier evaluation. *Applied Soft Computing*, 104:107219, 2021.

[215] Valery V Starovoitov and Yuliya I Golub. Comparative study of quality estimation of binary classification. In *Informatics*, volume 17, pages 87–101, 2020.

[216] W Nick Street and YongSeog Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 377–382, 2001.

[217] Andrés L Suárez-Cetrulo, David Quintana, and Alejandro Cervantes. A survey on machine learning for recurring concept drifting data streams. *Expert Systems with Applications*, 213:118934, 2023.

[218] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pages 843–852, 2017.

[219] C Szegedy. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[220] Xiangru Tang, Qiao Jin, Kunlun Zhu, Tongxin Yuan, Yichi Zhang, Wangchunshu Zhou, Meng Qu, Yilun Zhao, Jian Tang, Zhuosheng Zhang, et al. Prioritizing safeguarding over autonomy: Risks of llm agents for science. *arXiv preprint arXiv:2402.04247*, 2024.

[221] Mariusz Topolski and Jędrzej Kozal. Novel feature extraction method for signal analysis based on independent component analysis and wavelet transform. *Plos one*, 16(12):e0260764, 2021.

[222] Takumi Uchida and Kenichi Yoshida. Concept drift in japanese covid-19 infection data. *Procedia Computer Science*, 207:380–387, 2022.

[223] Muhammad Fahim Uddin. Addressing accuracy paradox using enhanced weighted performance metric in machine learning. In *2019 Sixth HCT Information Technology Trends (ITT)*, pages 319–324. IEEE, 2019.

[224] Gido M Van de Ven, Tinne Tuytelaars, and Andreas S Tolias. Three types of incremental learning. *Nature Machine Intelligence*, 4(12):1185–1197, 2022.

[225] Jan N van Rijn, Geoffrey Holmes, Bernhard Pfahringer, and Joaquin Vanschoren. The online performance estimation framework: heterogeneous ensemble learning for data streams. *Machine Learning*, 107:149–176, 2018.

[226] Joaquin Vanschoren. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*, 2018.

[227] Michel Verleysen and Damien François. The curse of dimensionality in data mining and time series prediction. In *International work-conference on artificial neural networks*, pages 758–770. Springer, 2005.

[228] Anna Vettoruzzo, Mohamed-Rafik Bouguelia, Joaquin Vanschoren, Thorsteinn Rognvaldsson, and KC Santosh. Advances and challenges in meta-learning: A technical review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[229] Haixun Wang, Wei Fan, Philip S Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 226–235, 2003.

[230] Geoffrey Webb, Loong Lee, François Petitjean, and Bart Goethals. Understanding concept drift. 04 2017.

[231] Geoffrey I Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. Characterizing concept drift. *Data Mining and Knowledge Discovery*, 30(4):964–994, 2016.

[232] Weronika Węgier, Michał Koziarski, and Michał Woźniak. Multicriteria classifier ensemble learning for imbalanced data. *IEEE Access*, 10:16807–16818, 2022.

[233] Peter Welinder, Steve Branson, Pietro Perona, and Serge Belongie. The multidimensional wisdom of crowds. *Advances in neural information processing systems*, 23, 2010.

[234] Norbert Wiener. *Cybernetics or Control and Communication in the Animal and the Machine*. MIT press, 1948.

[235] David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

[236] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[237] Kevin Woods, W. Philip Kegelmeyer, and Kevin Bowyer. Combination of multiple classifiers using local accuracy estimates. *IEEE transactions on pattern analysis and machine intelligence*, 19(4):405–410, 1997.

[238] Michał Woźniak. Experiments with trained and untrained fusers. *Innovations in hybrid intelligent systems*, pages 144–150, 2007.

[239] Michał Woźniak. Application of combined classifiers to data stream classification. In *IFIP International Conference on Computer Information Systems and Industrial Management*, pages 13–23. Springer, 2013.

[240] Michał Woźniak. *Hybrid classifiers: methods of data, knowledge, and classifier combination*, volume 519. Springer, 2013.

[241] Michał Woźniak, Manuel Grana, and Emilio Corchado. A survey of multiple classifier systems as hybrid systems. *Information Fusion*, 16:3–17, 2014.

[242] Michał Woźniak and Konrad Jackowski. Some remarks on chosen methods of classifier fusion based on weighted voting. In *Hybrid Artificial Intelligence Systems: 4th International Conference, HAIS 2009, Salamanca, Spain, June 10-12, 2009. Proceedings 4*, pages 541–548. Springer, 2009.

[243] Michał Woźniak and Andrzej Kasprzak. Data stream classification using classifier ensemble. *Schedae Informaticae*, 23:21–32, 2014.

[244] Michał Woźniak and Marcin Zmyślony. Designing fusers on the basis of discriminants–evolutionary and neural methods of training. In *Hybrid Artificial Intelligence Systems: 5th International Conference, HAIS 2010, San Sebastián, Spain, June 23-25, 2010. Proceedings, Part I 5*, pages 590–597. Springer, 2010.

[245] Feiyu Xu, Hans Uszkoreit, Yangzhou Du, Wei Fan, Dongyan Zhao, and Jun Zhu. Explainable ai: A brief survey on history, research areas, approaches and challenges. In *Natural language processing and Chinese computing: 8th cCF international conference, NLPCC 2019, dunhuang, China, October 9–14, 2019, proceedings, part II 8*, pages 563–574. Springer, 2019.

[246] Zhiwen Yu, Daxing Wang, Zhuoxiong Zhao, CL Philip Chen, Jane You, Hau-San Wong, and Jun Zhang. Hybrid incremental ensemble learning for noisy real-world data classification. *IEEE Transactions on Cybernetics*, 49(2):403–416, 2017.

[247] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

[248] Hongyu Zhu and Xizhao Wang. A cost-sensitive semi-supervised learning model based on uncertainty. *Neurocomputing*, 251:106–114, 2017.

[249] Paweł Zyblewski, Paweł Ksieniewicz, and Michał Woźniak. Classifier selection for highly imbalanced data streams with minority driven ensemble. In *International conference on artificial intelligence and soft computing*, pages 626–635. Springer, 2019.

[250] Paweł Zyblewski, Robert Sabourin, and Michał Woźniak. Preprocessed dynamic classifier ensemble selection for highly imbalanced drifted data streams. *Information Fusion*, 66:138–154, 2021.

[251] Paweł Zyblewski and Michał Woźniak. Novel clustering-based pruning algorithms. *Pattern Analysis and Applications*, 23(3):1049–1058, 2020.