technische universität
dortmund

Bachelor's Thesis

Online Adaptive Multivariate Time Series Forecasting

Hanna Mykula

October 2021

Supervisors:
Prof. Dr. Katharina Morik
Matthias Jakobs

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für künstliche Intelligenz (LS 8)
https://www-ai.cs.tu-dortmund.de/index.html

Contents

Chapter 1

Introduction

Time series forecasting plays an important role in strategic decision-making in a wide range of industries, e.g., economics, retail, business, finances, environmental and social studies, etc. Effective forecasting methods may provide meaningful information about the future based on given data. Various machine learning techniques have already been applied to solve real-world forecasting problems on data collected over time.

In a rapidly evolving digital environment, the representation of data often involves multiple inter-dependent variables. On the one hand, this may give data engineers additional information about the target concept. On the other hand, the dataset might include irrelevant and redundant features, which increase the dimensionality of the machine learning model input without contributing to the model's forecasting accuracy. Moreover, the changing and non-stationary nature of data distributions leads to a phenomenon of so-called concept drift. In order to reduce the generalization error and increase the efficiency of time series forecasting, we devise an online adaptive framework, which performs time series selection on the raw time series data and adapts to concept drifts by informed re-training.

1.1 Motivation and aim

Most of the existing time series forecasting models operate in a static manner, i.e. the model is trained offline using some historical data. Its parameters are optimized once at training time and the model is deployed in production with fixed learned parameters and fixed information about temporal and spatial data. However, in a rapidly evolving non-stationary environment, the relation between the input data and the target variable might change over time. This change is formally described as concept drift. As a result, previously learned concepts about data become no longer valid, making the offline model not appropriate for future predictions. For instance, the launch of new products on the market leads to a change in consumers' behavior and, consequently, concept drift in sales prediction model.

This thesis proposes an online time series forecasting method that uses dynamic time series selection and concept drift detection on given data. Dynamic time series selection ensures, that at each timestamp the model is being trained only on the relevant data, thus, avoiding the curse of dimensionality. This step consisted of two parts: selection of the relevant time series, exclusion of the redundant time series. We have compared the results of applying various distance and similarity measures to determine relevant features. To remove the redundancies from the selected subset, the relevant time series are clustered and only cluster representatives are selected. We have explored and compared different clustering strategies: k-means clustering with Euclidean distance and dynamic time warping clustering.

The main aspect of this work is concept drift detection within both temporal and spatial dimensions of the multivariate time series data. Spatial dependencies indicate the similarity between the values at one time instant. We investigate the change of the similarity measures between the two most dissimilar processes using Hoeffding Bound. Temporal dependencies indicate the patterns discovered within the same dimension. The drift detection within the temporal dimension is ensured by tracking the change in estimated model's performance error over time.

1.2 Structure of the thesis

This work is structured as follows. Section 2 gives an overview of the key terms needed in the time series and concept drift detection fields, introduces the formulation of the problem. Section 3 describes the state-of-the-art approaches and related work on concept drift detection and adaptation. Section 4 gives a detailed description of the statistical and machine learning models, to which we applied our framework. Section 5 presents our online adaptive time series forecasting framework. We evaluate our approach by comparing it to standard baselines methods for time series forecasting in section 6. The thesis concludes with a summary and a short discussion in section 7.

Chapter 2

Key Terms and Problem Formulation

Before introducing the phenomenon of concept drift and our online adaptive forecasting framework, we will first focus on the definition of multivariate time series.

The series of data points indexed in time order form a time series. However, some real-life scenarios involve multiple inter-dependent variables. One example of a possible scenario is weather forecasting. We require complementary variables like humidity, precipitation percent, wind speed, etc., in addition to temperature, to be able to optimally predict the temperature in the future. A statistical model that deals with dependent data over time is denoted as multivariate time series forecasting model. Multivariate time series analysis considers the discovery of repeating patterns within two dimensions: spatial and temporal. We will introduce the formal notation of spatial and temporal dimensions in the chapter below.

2.1 Time Series

The definition of time series was generalized by the *value series* by Morik and Mierswa in [23]. They define each element y_i of the series as an instance consisting of the index, indicating unit of time, and an m-dimensional value vector. The value vector is an element of the value space.

2.1.1 Definition. (Value Series) A value series is a mapping $y : \mathbb{N} -> \mathbb{R} \times \mathbb{C}^m$. y_n is the element of series with index n and $(y_i)_{i \in \{1,..,n\}}$ a series of length n.

The introduction of the index component allows equally as well as unequally spaced value series, while the complex number value space grants a convenient way to use basis transformations like the Fourier transformation. For the reasons named above, the definition of value series covers both time series and their transformations. However, the target of the following work is multivariate time series.

2.1.2 Definition. (Multivariate Time Series) A time series is a set of observations, $Y^i = \{y^i_1, y^i_2 ..., y^i_t\}$ made sequentially through time, where y^i_j indexes the value of time series i at time j; $y^i_j \in \mathbb{R}, \forall j \in \{1, 2, ..., t\}$. If i is greater than 2, the time series is denoted as multivariate time series.

Multivariate time series $Y = \{y_1, y_2, ...y_t\}$ can be represented as follows:

$$\mathbb{Y} = \begin{pmatrix} y^1_1 & \cdots & y^1_j & \cdots & y^1_t \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ y^i_1 & \cdots & y^i_j & \cdots & y^i_t \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ y^M_1 & \cdots & y^M_j & \cdots & y^M_t \\ \end{pmatrix}$$

where vector $y^i = (y^i_1, ...y^i_j, ...y^i_t)$ is called a value vector of time series with index i and will further be denoted as *feature*. The value vector y^i describes the *temporal* dimension of mulivariate time series y. A vector $y_j = (y^1_j, ...y^i_j, ...y^M_j)$ is a vector of y at time j, which characterizes the *spatial* dimension of multivariate time series. M and t are the sizes of spatial and temporal dimensions respectively.

2.1.3 Definition. (Feature Space) Given time series $Y = \{y_1, y_2, ..., y_t\}$, feature space $X \subset R^k$ is defined by

$$x_i = (y_{i-k}, y_{i-k+1}, ...y_i), x_i \in X,$$

where k is a lag, which specifies the number of past observations to be considered in feature space.

The standard approaches to handle time series include classic **linear** statistical models, such as Autoregressive Integrated Moving Average (ARIMA) for univariate time series and Vector Autoregression (VAR) for multivariate. However, such models may be insufficient in practice due to the non-linearity of real-life time series scenarios. Hence, various machine learning techniques have already been applied to time series forecasting problems to perform non-linear modeling. But before moving to the role of machine learning in the time series domain, we will first define what is a forecasting task.

2.1.4 Definition. (Forecasting) Given a time series matrix Y defined in 2.1.2 and constructed feature space X, the task is to predict the value y_{t+1}, using past k observations:

$$y_{t+1} = f(x_t), \text{ for } x_t \in X$$

where f is the prediction function learned by the model.

2.1.5 Definition. (Supervised Machine Learning) Supervised Learning is a machine learning task, where the model learns a function that maps an input to an output based on example input-output pairs: (X_i, Y_i), for X_i being an input feature vector and Y_i being an expected output at the time i.

Supervised learning algorithms are trained using labeled examples, such as input values where the desired output is known. In other words, labeled instances are defined as a set of data assigned with some meaningful and informative "tag" (e.g. what does the photo contain, "spam" flag on email messages, etc.). Learning is achieved by comparing its actual output with correct outputs in order to find errors and modifying the model according to the results.

Supervised learning can be classified into two major prediction techniques: *classification* and *regression*. Classification algorithms categorize data under different labels according to a given input outputting a discrete value. Regression algorithms look for numerical (or continuous) values. In the case of regression, the output is the real value, which can be an integer or floating value (e.g. quantities, sizes, etc.).

2.1.6 Definition. (Regression) Given a time series matrix Y defined in 2.1.2 and the task to predict the value y_{t+1}. The goal of the regression is to approximate function $f : \mathbb{X} \rightarrow \mathbb{Y}$, where $\mathbb{X}$ is a constructed feature space, so that

$$y_{t+1} = f(x_t), \text{ for } x_t \in X$$

is as close as possible to the real value.

The construction of multivariate feature space is also denoted as *time series embedding.* An embedding is a process of translating high-dimensional vector in a relatively low-dimensional space. It reduces the dimensionality of the input data, while still retaining the important information about the target variable. It reformulates the time series forecasting task into a supervised learning regression task. Time series embedding was formally defined in [8].

2.1.7 Definition. (Time Series Embedding) An embedding of time series Y into embedding space of size k is a mapping $\mathbb{R}^t \rightarrow \mathbb{R}^k$, which constructs a multivariate feature space:

$$X_{t-k} = \{x_1, x_2, ...x_{t-k}\},$$

where x_i is a feature space of time series i.

In this work, we perform a two-stage time series selection on the multivariate time series data. The first step is similarity-based selection. We have applied and compared different measures to estimate the most coherent time series with the target one, such as similarity measures based on Euclidean, Fourier Coefficient and Manhattan distances, as well as Pearson and Spearman correlations. Distance between target time series y^p and candidate time series y^q is measured as follows:

2.1.8 Definition. (Manhattan Distance)

$$d(y^p, y^q) = \sum_{i=1}^{n} |y_i^p - y_i^q|$$

2.1.9 Definition. (Euclidean Distance)

$$d(y^p, y^q) = \sqrt{\sum_{i=1}^{n} (y_i^p - y_i^q)^2}$$

where y^p and y^q are the value vectors of time series with index p and q respectively and n is the temporal dimension of the time series. This two measures show very similar behaviors on different kinds of data. This is explained by that fact that both measures compute the difference in time series values. However, Euclidean distance is more sensitive to outliers than Manhattan distance due to its non-linear characteristics [1].

2.1.10 Definition. (Fourier Coefficient based Distance) Fourier coefficient-based distance is the Euclidean distance between two time series y^p and y^q with Fourier coefficients $y_f^p = \langle (m_0, l_0), \cdots (m_{n-1}, l_{n-1}) \rangle$ and $y_f^q = \langle (r_0, s_0), \cdots (r_{n-1}, s_{n-1}) \rangle$:

$$d(y^p, y^q) = \sum_{i=1}^{n-1} (m_i - r_i)^2 + (l_i - s_i)^2$$

Fourier-based similarity measure is highly sensitive to scaling and noise effects than Euclidean and Manhattan distances, but more robust to amplitude changes [1].

2.1.11 Definition. (Pearson Correlation)

$$r(y^p, y^q) = \frac{\sum_{i=1}^{n} (y_i^q - \overline{y^q})(y_i^p - \overline{y^p})}{\sqrt{\sum_{i=1}^{n} (y_i^q - \overline{y^q}) \sum_{i=1}^{n} (y_i^p - \overline{y^p})}}$$

2.1.12 Definition. (Spearman Correlation) The Spearman correlation coefficient is defined as the Pearson correlation coefficient between the rank variables. The rank $R(x_i)$ is the position of numerical value x_i in the sorted set of data x.

$$r(y^p, y^q) = \frac{\sum_{i=1}^{n} (R(y_i^q) - R(\overline{y^q}))(R(y_i^p) - R(\overline{y^p}))}{\sqrt{\sum_{i=1}^{n} (R(y_i^q) - R(\overline{y^q})) \sum_{i=1}^{n} (R(y_i^p) - R(\overline{y^p}))}}$$

The main difference between two correlations measures is that Pearson correlation measures the linear relationship between two time series, while Spearman correlation only measures monotonic one.

The second step of the framework includes the exclusion of the redundant features by time series clustering to ensure the diversity of the input data.

Clustering is the partitioning of multiple univariate time series into groups based on their similarity measure [17]. The desired property of every clustering algorithm is, that all members of the same cluster are similar to each other, but are as dissimilar as possible from objects in a different cluster. This problem is formally defined as follow [2]:

2.1.13 Definition. (Time Series Clustering) Given a set of n univariate time series $Y = \{y_1, y_2, ...y_n\}$, time series clustering partitions Y into $C = \{C1, C2, ...C_k\}$ in such way, that homogenous time series are grouped together based on chosen similarity measure. The C_i is called a cluster, where $Y = \cup_{i=1}^{k} C_i$ and $C_i \cap C_j = \emptyset$ for $i \neq j$.

2.1.14 Definition. (K-means Clustering) K-means clustering is one kind of centroid based clustering. In centroid-based clustering the cluster center is defined by the average of all cluster members. The candidate time series is the one assigned to the closest centroid. The K stays for the number of clusters. The initialization of centroids is done by randomly selecting K time series for the cluster centers.

We compared k-means clustering using different distance measures: Euclidean distance (see 2.1.9) and dynamic time warping. Dynamic time warping algorithm is the time series similarity measure, which overcomes the restrictions of Euclidean distance. It allows to calculate matching distance by minimizing a cumulative distance measure of distances between samples of one time series [26]. Dynamic time warping distance problem is solved by using a dynamic programming approach.

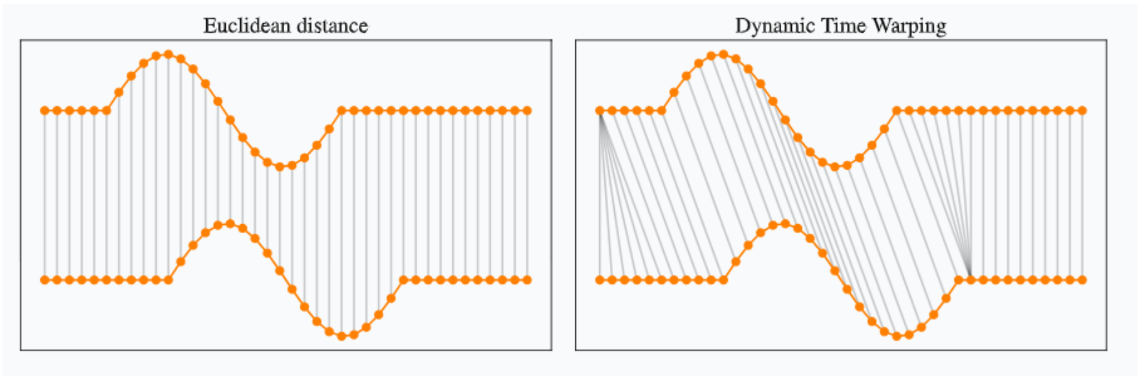

Figure 2.1: Euclidean vs. DTW Distances. `https://rtavenar.github.io/blog/dtw.html`

2.1.15 Definition. (Dynamic Time Warping) The DTW-distance between two time series $y^p = (y^p_1, ...y^p_{t_0})$ and $y^q = (y^q_1, ...y^q_{t_1})$ of length t_0 and t_1 respectively is:

$$D(i,j) = min \left\{ \begin{array}{l} D(i,j-1) \\ D(i-1,j) \\ D(i-1,j-1) \end{array} \right\} + d(x^p_i, x^q_j),$$

where i starts with t_0 and j with t_1.

The time series analysis requires the modelling of the underlying stochastic process that forms the time series.

2.1.16 Definition. (Stochastic Process) A model that describes the probability structure of a sequence of observations is called a *stochastic process* [31]. A stochastic process is a collection of random variables X_t indexed by a set T, i.e. $t \in T$. Such a random process can be used to described by the moments of its time samples:

- The *mean* of a random process X_t for time t is defined as follows
$$\mu(t) = E[X(t)]$$
- The *variance* of a random process X_t for time t is defined by
$$\sigma^2 = VAR[X(t)]$$
- The *autocovariance* of a random process X_t for two time moments t_1 and t_2 is defined by
$$\gamma(t_1, t_2) = E[(X(t_1) - \mu(t_1))(X(t_2) - \mu(t_2))]$$

2.1.17 Definition. (Stationary process) Stochastic process is considered stationary, if *mean*, *variance* and *autocovariance* do not change when shifted in time. A k-dimensional time series x_t is strictly stationary if the joint distribution of the m collection, $(x_{t_1},...,x_{t_m})$, is the same as that of $(x_{t_{1+j}},...,x_{t_{m+j}})$, where m, j, and $(t_1, ..., t_m)$ are arbitrary positive integers [30].

As stated in the introduction, data is expected to change over time, especially in environments where the change in the underlying random process over time is common. Consequently, the moments of the stochastic process significantly diverge as time passes by. Such processes are denoted as *non-stationary.* In order to simplify the potential solution search and reduce the time invested in finding the alternative solution, the stationarity assumptions are rarely being tested in practice. Most of the existing methods in applied time series analysis are based on the assumption, that the underlying probability distribution is constant.

However, one recent work [24] has claimed, that the assumptions of the existence of a stationary distribution cannot be taken for sure. In some real-life scenarios, the periods of change in the underlying concept might be of paramount importance for business and industry decision-making.

2.2 Concept Drift

As already mentioned above, the main aspect of this work is the concept drift detection and adaptation mechanism. In this section, we will discuss the possible nature and kinds of concept drifts in the data.

2.2.1 Definitions

The main characteristic of the dynamically changing environment is, that the change usually happens unexpectedly. Furthermore, the change might have different nature of origin, e.g., the change in the input data or the change in the learned relation between input features and the target variable.

2.2.1 Definition. (Concept Drift) The formal definition of *concept drift* between time point t_0 and time point t_1 can be expressed as

$$\exists X : p_{t_0}(X, y) \neq p_{t_1}(X, y),$$

where p_{t_0} denotes the joint distribution at time t_0 between the set of input variables X and the target variable y [11]. The change in data between two points in time t_1 and t_0 can be explained by the possible changes in the components of this relation [16, 22]. That is, the cause of the concept drift may be the change in one of the three probabilities $p(y)$, $p(X|y)$ or $p(y|X)$.

2.2.2 Concept Drifts types

Based on the origin of the change, we can categorize all concept drifts into two groups:

1. *Real drift* indicates the change in the posterior probabilities $p(y|X)$. Real concept drift may also happen at the same time with the change in the input data characteristics $p(X)$.

2. *Virtual drift* occurs if the change in the distribution of incoming input data $p(X)$ has happened, but without any impact on $p(y|X)$.

We can distinguish between different structural forms of change in the data distribution over time. The main forms of concept drift are illustrated on the Figure 2.2. In this example, we consider one-dimensional data, where the origin of drift is defined by the mean of the data.

A *sudden/abrupt* drift happens, when suddenly replacing one concept with another one. A drift happens *incrementally* or *gradually*, when many intermediate concepts exist between the old and new concept. However, it is important to observe such kinds of drifts on a longer time interval, as the difference between intermediate concepts is expected to be very small. Another type of concept drifts are *reoccurring contexts*. They can be explained by seasonal patterns, i.e. when a previously valid concept repeats after some time.

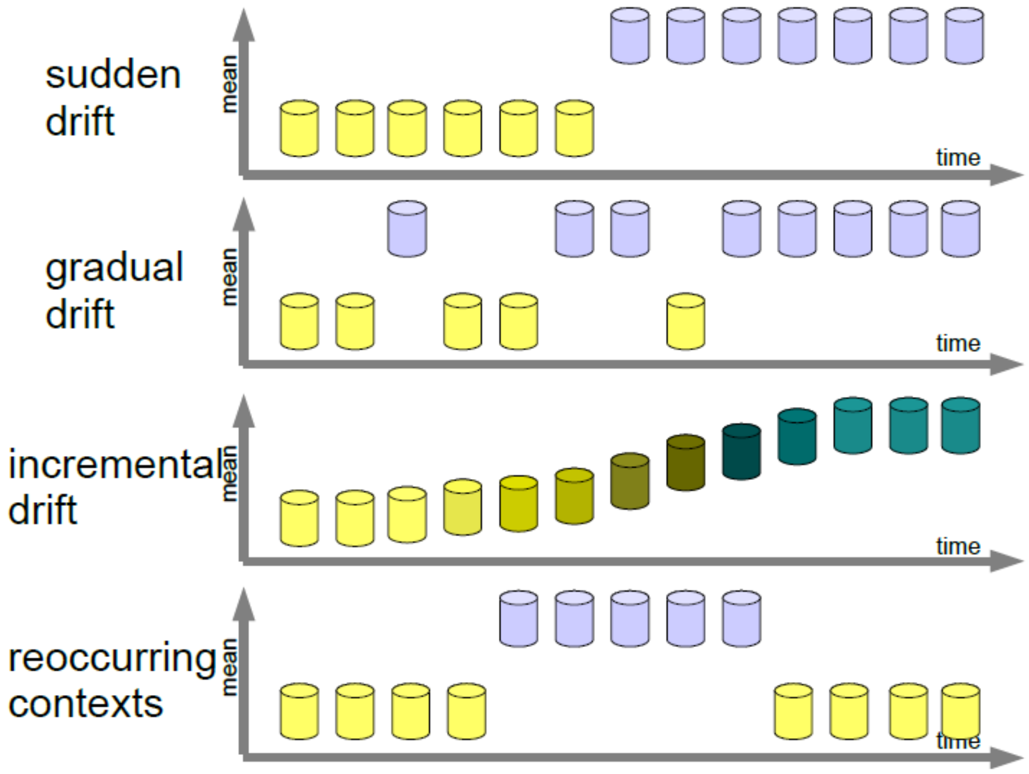

Figure 2.2: Forms of Concept Drift [21].

2.3 Evaluating forecast accuracy

It is common to split the data into **train** and **test** data. The training data is used for estimating the parameters of the forecasting model, while test data is used to evaluate its accuracy. Given a set of the forecast $\hat{Y}_m = \{\hat{y}_{t+1}, \hat{y}_{t+1}, ..., \hat{y}_{t+n}\}$ and a set corresponding observed values $Y_m = \{y_{t+1}, y_{t+2}, ..., y_{t+n}\}$, where m is the candidate model and n is the size of test set, we compute two most popular accuracy measures: Root mean squared error (RMSE) and Symmetric mean absolute percentage error (SMAPE).

2.3.1 Definition. (Root mean squared error)

$$RMSE = \frac{1}{n}\sqrt{\sum_{t=1}^{n}(y_t - \hat{y}_t)^2}$$

The RMSE error value is in the same units as the forecast. We compute the Root mean squared percentage error (RMSPE) as follows:

$$RMSPE = \frac{RMSE}{mean(Y_t)}$$

The RMSE is a scale-dependent measure, that is why we cannot use RMSE to compare the accuracy of two different time series forecasts with different measure units.

2.3.2 Definition. (Symmetric mean absolute percentage error)

$$SMAPE = \frac{1}{n}\sum_{t=1}^{n} \frac{|\hat{y}_t - y_t|}{(y_t + \hat{y}_t)/2}$$

The SMAPE is a scale-independent error metric. However, it has one restriction: the values $\hat{Y}_t$ and Y_t should be positive.

2.4 Online vs. Offline Learning Tasks

We can categorize the learning tasks into two groups: *offline* and *online* learning. Offline learning refers to learning when the training data is available only once, at the training time. After completing the training, the model can be deployed in production and used for training. In offline learning, the prediction function $f(.)$ is kept constant over time.

Online learning refers to the learning when the training data is being processed sequentially, i.e. one data point at a time. The prediction function $f(.)$ can be updated, as more training data is being discovered over time. The framework proposed in this thesis operates on the multivariate time series regression problems in an online fashion.

2.5 Problem Definition

The problem considered in this thesis is to forecast multivariate time series. Let $MTS = \{ts_1, ts_2, ..., ts_m\}$ be the set of m time series, i.e. $\mathbb{M} = \{1, ..., m\}$, while $Y^i = \{y^i_1, y^i_2 ..., y^i_t\}$ denote a time series, where $y^i_t \in \mathbb{N}$ represents the value of time series i at time instant t. Formally, each $Y^i, \forall i \in \mathbb{M}$ is an infinite sequence of random variables forming a random process, where y^i_t is the random variable at instant t. Let $\mathbb{Y}$ represent a $M-$dimensional vector of the random processes $\{Y^1, Y^2, ..., Y^M\}$. k^i is the number of considered lags in each ts Y^i. Based on the above notation, it is possible to formulate the multivariate time series forecasting problem as follows.

2.5.1 Definition. (Online Multivariate Time Series Forecasting Problem).

Given: Historical realizations of random demand processes $Y^i, \forall i \in \mathbb{K}$ and $\mathbb{Y}$ denoting the following Euclidean space:

$$\mathbb{Y} = \begin{pmatrix} y^1_1 & \cdots & y^1_{t-k^1+1} & \cdots & y^1_t \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ y^i_1 & \cdots & y^i_{t-k^i+1} & \cdots & y^i_t \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ y^M_1 & \cdots & y^M_{t-k^M+1} & \cdots & y^M_t \end{pmatrix} = \begin{pmatrix} \mathbf{Y}_1 \\ \vdots \\ \mathbf{Y}_{t-j} \\ \vdots \\ \mathbf{Y}_t \end{pmatrix}^T$$

Objective: Forecast all the K realizations of $\mathbb{Y}$ at instant $t+1$.

Most of the existing approaches to forecast time series are designed unaware of concept drifts. However, as already mentioned in 2.1, the assumptions about the stationarity of time series do not always hold. The relation in the input dependencies and the performance of the model changes over time. Thus, the need to update the model emerges. One of the methods to handle concept drift is to periodically retrain the model, independently from the concept drift occurrence. However, such an approach can potentially lead to extreme consumption of computational resources and long processing times. In this thesis, we present a two-stage concept drift detection mechanism with dynamic feature selection. In the first stage, we compute the similarity between the features of the raw data over time and compare if the similarity measure of the two most dissimilar processes significantly diverges as more input data arrives using Hoeffding bound. In the second stage, we compute the estimated forecast error of the candidate model and carry out the Page-Hinkley test to estimate whether there is a drastic negative change in the models' accuracy.

Chapter 3

Related Work

In this chapter, we introduce related works and existing approaches that are used to detect and handle concept drift on multivariate time series data. Concept drift detection and adaption done in the domain of the classification problems has been researched in bigger depth compared to the domain of regression problems.

The existing methods proposed for handling concept drifts can be categorized into two main groups: (1) informed and (2) uninformed methods. Uninformed drift detection methods [20] blindly retrain the predictive model in a fixed period of time without any information about characteristics of concept drift. However, the possible resource consumption, which might not contribute to the model's predictive quality, is an unavoidable consequence of periodic retraining in an uninformed way, when choosing a small interval between two trainings. If the interval is, in opposite, too big, the model might not adapt to drift at all.

Informed or white box drift detection uses some statistical information about the data or model performance to inform the model about the occurrence of concept drift and, if necessary, update the predictor. We will present the main contributions and approaches for handling the concept drifts in both uninformed and informed manner in the section below.

3.1 Time Windowing for Concept Drift Detection

The approaches based on time windowing of fixed or adaptive size make the first group of informed drift detectors [12].

The main idea of such methods is to conduct a statistical test on the examples from the defined time window, which will make an assumption about a possible change in the underlying distribution. If it is the case, the model will then be retrained only on the newest examples included in the window. The main challenge of the proposed approach is to find the window of the optimal size (i.e. balancing between the relevance and the number of the training examples). The general rule of adaptive windowing is to decrease the window size, when the concept drift is assumed to take place and increase in the opposite case. One of the first adaptive windowing algorithms was FLORA2 [33]. Widmer and Kubat continuously monitored the accuracy and the coverage of the current model to **guess** the change in the underlying concept and adapted the window size according to the results of the statistical test.

One more example of handling concept drifts using adaptive windowing was developed by Klinkenberg and Joachims in [18]. In their method, they have used support vector machines as a base-learner. The main idea of the following approach can be broken down to the following optimization problem: adjust the window size so, that the estimated generalization error on new examples is minimized. The generalization error was measured based on ε_α method for support vector machines. However, this method is restricted to using a support vector machine and cannot be applied to every kind of forecasting problem. The hypothesis of time windowing is, that the most recent data is expected to be the most relevant and important for the target concept. Yet such assumption can not be taken, when the data is noisy or contains reoccurring concepts. Moreover, the window size might as well be shorter than the length of the change.

3.2 Example Weighting

The approach proposed by Klinkenberg and Joachims in [18] was extended in the following work [19] using a support vector machine as well. This work takes into account the importance of single data points by integrating a weighting scheme. The weights are assigned only to the examples from the most recent batch of data. They are selected based on their corresponding age, using an exponential aging function.

3.2.1 Definition. (Exponential Aging Function)

$$w_\lambda(x) = exp(-\lambda t_x),$$

where x is an example observed t_x time steps ago.

The key idea of this work is to slowly reduce the impact of uninformative examples on the learning result, instead of completely excluding them from the training. Unfortunately, the limitations named above apply to this framework too.

3.3 Concept Drift Detectors

Concept Drift Detector is a mechanism, that identifies the drift and change in the characteristics of the studied object. The advantage of such methods is that they are able to provide information about the nature and further characteristics of concept drift dynamics.

The basis for several change detection algorithms is the sequential probability ratio test [32]:

3.3.1 Definition. (Sequential Ratio Probability Test) Given a sequence of inputs X_1^n, the subset of inputs X_1^w, $1 < w < n$, which is generated from the unknown distribution P_0 and the subset of inputs X_w^n, which is generated from another unknown distribution P_1. A null hypothesis that there was a change in the distribution at time w is given by

$$T_w^n = \log \frac{P(x_w...x_n|P_1)}{P(x_w...x_n|P_0)} > L$$

where L is a user-defined threshold.

The most acknowledged technique based on the sequential probability ratio test is the cumulative sum (CUSUM) technique developed by E.S. Page [25]. It is often used for change detection and involves the calculation of the cumulative sum of residual, e.g. estimated prediction error of the model.

3.3.2 Definition. (CUSUM) Let x_t be the observed value (e.g. prediction error) at time t, δ be the allowed magnitude of changes and λ a user defined threshold. The CUSUM test is then defined by:

$$g_t = max(0, g_{t-1} + (x_t - \delta)), (g_0 = 0) \text{ for positive change}$$

$$g_t = min(0, g_{t-1} + (x_t - \delta)), (g_0 = 0) \text{ for negative change}$$

If g_t is greater (for positive change) or smaller (for negative change) than λ, then the drift is considered to take place and g_t will be set to 0.

One of the variants of the CUSUM algorithm is the Page-Hinkley (PH) Test [25]. This sequential analysis technique is common for drift detection in signal processing.

3.3.3 Definition. (Page-Hinkley Test) For the allowed magnitude of changes ν and user-defined threshold ϱ, the PH test is defined as follows:

$$m_t = \sum_{t=1}^{T} (x_t - \overline{x}_T - \nu)$$

$$M_T = min(m_t, t = 1...T)$$

$$PH_T = m_T - M_T$$

where $\overline{x}_T$ is the mean of the values observed up to time T. If the condition $PH_T > \varrho$ is fulfilled, then the drift alarm will be triggered.

3.4 Ensemble Methods

An ensemble is a set of different models that serve to predict the same target variable. Ensembles have been claimed to be a powerful method when it comes to time series forecasting. The general assumption behind, the combination of predictions returned by multiple weaker models improves the accuracy of the prediction.[14].

The work by Scholz and Klinkenberg [28, 29] introduces an ensemble method for handling concept drifts. The weights of previous models in the ensemble are updated based on their performance on the most recent batch of data. Formerly, the algorithm samples out the instances that were not classified right by the previous models, and induces the new models from the sampled examples. This method does not explicitly detect concept drifts, but adds more models to the ensemble, thus, needing more computational resources.

3.5 Evaluation of Drift Detection Methods

Another research [5] has focused on developing a framework to evaluate change detection methods. The motivation behind this work is to go beyond the classic error-based evaluation of drift detection methods for real-time predictions, as they might not be informative enough. The framework proposed in the work mentioned above is built as an extension of MOA: Massive Online Analysis. "Massive Online Analysis (MOA) is a software environment for implementing algorithms and running experiments for online learning from data streams" [4]. It uses the following criteria for an effective evaluation of change detection methods: Mean Time between False Alarms, Mean Time to Detection, Missed Detection Rule, Average Run Length [5].

Chapter 4

Forecasting

In this chapter, we present a pool of models used for time series forecasting to which we apply our framework. The models used for time series forecasting can be categorized into two groups: (1) statistical linear models and (2) non-linear machine learning models. The former group base on the theory of stochastic process and time series, while the latter one stems from machine learning study. First of all, we will introduce a statistical model used for multivariate time series forecasting, particularly Vector Autoregression. After that, we will look at the further machine learning models, which tended to show good performance on time series forecasting problems.

4.1 Vector Autoregression

The most widely used multivariate time series statistical model is the vector autoregressive model. It is called autoregressive model, because each value of the target variables is modelled as a function of its own past observations, i.e. the predictors can be described as time delayed value of the series. The main assumption is, that the prior values of the series are able to predict the present as well as the future. VAR models the k-dimensional random process with lag p as follows:

$$\hat{Y}_{t+1} = Y_t^T \beta_0 + ... + Y_{t-p+1}^T \beta_{p-1} + \epsilon_t = \sum_{j=1}^{p} Y_{t-j+1}^T \beta_{j-1} + \epsilon_t,$$

where $\beta = (\beta_0, \beta_1, ..., \beta p - 1) \in \mathbb{R}^{p \times k}$ are the parameters of the model. Thereby, ϵ_t is the vector of white noise, i.e. random process with zero mean and constant variance.

4.2 Projection Pursuit Regression

Projection Pursuit Regression is a linear model, which is induced as a sum of non-linear transformation of linear combination of the features. This method uses initially unknown smoothing functions f_j to estimate the variables from noisy scattered data.

$$y_i = \beta_0 + \sum_{j=1}^{r} f_j(\beta_j^T x_i)$$

where x_i is a feature space, β_j is a collection of r vectors containing unknown parameters and r is a hyperparameter.

4.3 Partial Least Squares Regression

Partial Least Squares Regression is a statistical technique, which breaks the predictors down to a smaller set of independent components and performs least squares regression on them.

4.3.1 Definition. (Least Squares Regression) Least Square Regression is a linear model, which aims to find model parameters that best fit the data. Its main goal is to minimize the sum of squared residuals S:

$$S = \sum_{i=1}^{n} (y_i - f(x_i, \beta))$$

4.4 Multivariate Adaptive Regression Splines

Multivariate adaptive regression splines (MARS) is an algorithm, that handles nonlinearities between features by using nonparametric regression [9]

$$f(x) = \sum_{m=1}^{M} a_m B_m(x)$$

where B_m is the basis function and c is a constant coefficient. Each basis function can be one of the following:

1. a constant 1.
2. a function of form $\max(0, x - \text{constant})$ or $\max(0, \text{constant} - x)$
3. a product of two or more functions from (2).

MARS is a two-stage algorithm. It consists of the forward and the backward pass. In the first phase, MARS aims to iteratively add the pair of basis functions that achieve the maximum reduction in sum-of-squares residual error. This phase usually overfits the model. In the second phase, the backward pass, the algorithm removes the terms added in the first phase one by one, keeping only the most effective terms. The algorithm stops as soon as the best submodel is found. The backward pass uses generalized cross validation to compare the accuracy of separate subsets of terms in order to find the best one.

4.5 Gradient Boosting Machine

Gradient Boosting Machine is a prediction model developed in the form of an ensemble of weak predictors, based on the same method. The typical baseline model is a decision tree. Gradient indicates the error, or residual, computed after building a model. The boosting algorithms aim to incrementally increase the model's accuracy. Given a set of training pairs (x_i, y_i), $i \in \{1, ..., n\}$, a solution of gradient boosting machine is defined as a sum of series of models, each trained on a different training set, to produce a final prediction

$$F(x) = \sum_{m=0}^{M} f_m(x),$$

where M is the amount of iterations and $f_m(x)$ are incremental functions, or "boosts", induced by the optimization algorithm ($f_0(x)$ is the initial model, trained on the training set). The "boost" $f_m(x)$ is then defined by

$$f_m(x) = -\rho_m g_m(x),$$

where gradient of the model and current step size for the model are calculated correspondingly

$$g_m(x) = \left[\frac{\theta E_y[L(y,(F(x))|x]}{\theta F(x)}\right]_{F(x)=F_{m-1}(x)}$$

$$\rho_m = \text{argmin} E_{y,x} L(y, F_{m-1}(x) - \rho g_m(x))$$

with $L(y, F(x))$ being a loss function. The gradient descent of the current model are the targets, that are used to train a new model, i.e. the new model f_m is inferred using an updated dataset of $\{(x_i, g_m(x_i))\}$. More on the gradient boosting machines theory can be found in [10].

4.6 Random Forest

Random Forest [6] is an ensemble learning method for classification and regression, which consists of multiple decision trees constructed at training time on different subsets of data. The basis of this learning algorithm is a method denoted as bootstrap aggregation, or bagging [7]. At each tree split in the learning process, the procedure extracts a random subset with the replacement from the dataset and trains the next predictor on it. The key idea behind the random forests is to obtain the set of models that are trained on different underlying patterns. The final prediction is induced by the average of all trees in the model. Bagging helps to generalize the prediction and, thus, lower the model's variance.

4.7 Support Vector Regression Machine

Support vector regression machine is supervised machine learning algorithm used for solving regression problems, based on the idea of support vectors for classification problems. Support vector regression can be formulated as following optimization problem: define and minimize a convex ε-insensitive loss function, find the flattest tube that covers most of the training instances [3]. One possible visualization of the support vector regression is represented below.

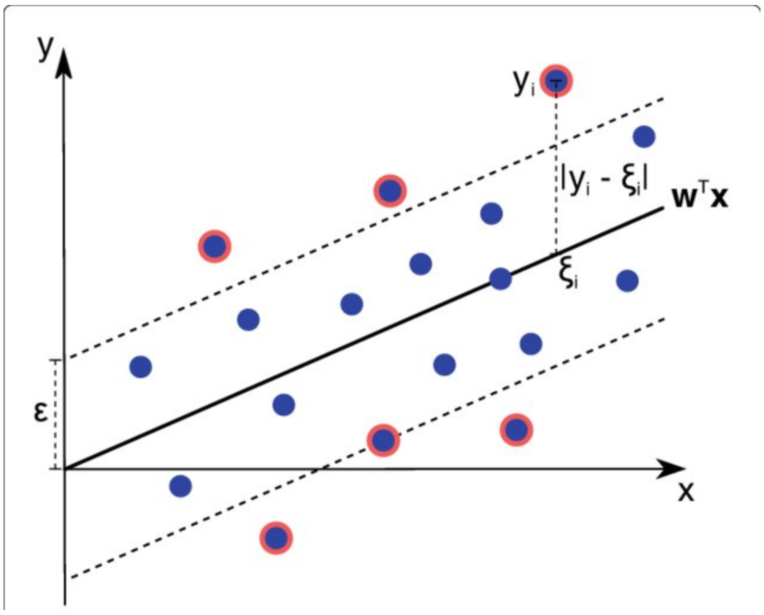

Figure 4.1: Support Vector Regression. [27]

The function $f(x)$ being approximated for M-dimensional multivariate time series can be defined as

$$f(x) = \langle w, x \rangle + b, b \in \mathbb{R}, w, x \in \mathbb{R}^M$$

As mentioned above, one of the tasks of support vector regression to find the narrowest tube centered around the hyperplane, while minimizing the estimated prediction error. This can be achieved by minimizing the norm, i.e. $\|w\|^2$. This problem can be devised as following convex optimization problem:

$$\begin{aligned} &\text{minimize } \frac{1}{2}\|w\|^2 \\ &\text{subject to} \begin{cases} y_i - \langle w, x_i \rangle - b \leqslant \varepsilon \\ \langle w, x_i \rangle + b - y_i \leqslant \varepsilon \end{cases} \end{aligned}$$

However, the important step to handle non-linearity using support vector regression is done by using kernels. Kernel is a function $\Phi : \chi \rightarrow \mathcal{F}$, that maps training instances into some feature space $\mathcal{F}$. In this work we compared following kernels [3]

$$K(w, x) = \begin{cases} \exp(-\frac{\|x - w\|}{\sigma}) & \text{laplacian} \\ (\alpha x^T w + c)^q, q > 0 & \text{polynomial} \\ \exp(-\frac{\|x - w\|^2}{\sigma^2}) & \text{rbf} \\ \langle w, x \rangle & \text{linear} \end{cases}$$

α, σ, c are user-defined parameters. The approximated function $f(x)$, which uses kernel, looks as follow

$$f(x) = K(w, x) + b, b \in \mathbb{R}, w, x \in \mathbb{R}^M$$

4.8 Deep Neural Networks

Deep Neural Networks have been considered to be one of the most robust and capable tools when it comes to a huge amount of non-linear data. Neural Network is the base of all deep learning algorithms, which was inspired by the nature of the human brain. It is structured as multiple layers of neurons. The neurons, or nodes, of one layer are connected with the neurons of the next layers. Neural Network usually consists of an input layer, one or more hidden layers and an output layer. An output layer can contain one or more nodes, for regression problems it contains a single neuron. We can think of each neuron as a separate linear regression model, producing an output y:

$$y = g(\sum_{i=1}^{n} w_i x_i + b),$$

where x_i is an input, w_i is weight, b is a bias(or threshold) and g is an activation function. The activation function g ensures, that the neural network is able to handle non-linear character of data. The most common activation functions are sigmoid, hyperbolic tangent and rectified linear unit (ReLU) functions:

$$g(x) = \begin{cases} \dfrac{1}{1+e^{-x}} & \text{sigmoid} \\ \dfrac{2}{1+e^{-2x}} - 1 & \text{tahn} \\ \begin{cases} 0, & \text{for } x < 0 \\ x, & \text{for } x \geqslant 0 \end{cases} & \text{ReLU} \end{cases}$$

The weights and biases are the learned parameters of the model. The weight gives in the importance of the given input variable. Neural networks use **backpropagation** to tune the weights based on the estimated error rate achieved in the previous training iteration. Backpropagation computes the gradient of the loss function to provide the level of adjustment for network's learnable parameters. Mathematically, the gradient is a partial derivative of the loss function L with respect to the learned weight w:

$$\frac{\partial L}{\partial w}$$

The gradient defines to which extent the parameter w has to be adjusted in order to minimize the loss function L.

4.8.1 Multilayer Perceptron

The most basic example of neural networks is a multilayer perceptron (MLP). It is composed of an input layer to process the input, an output layer to make a prediction and a arbitrary number of hidden layers, which act as a computational engine. MLPs are usually described as fully connected networks, i.e. each node in one layer is connected to all nodes in the next layer.

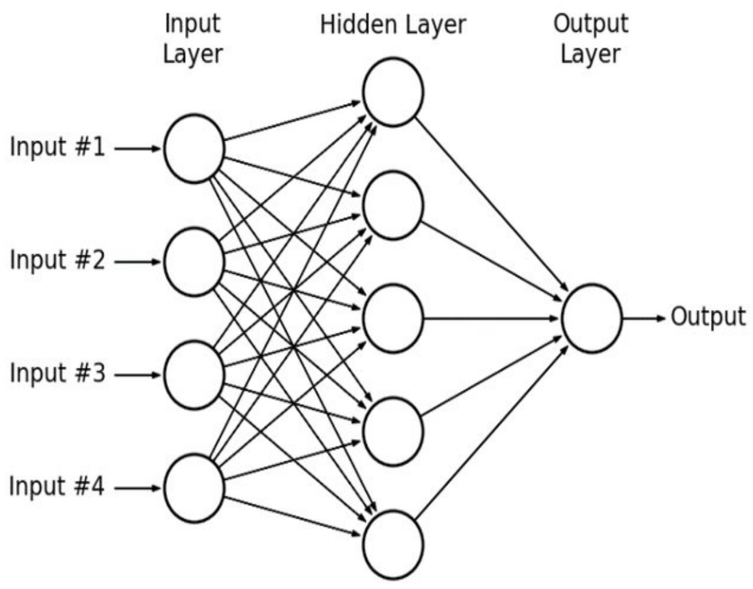

Figure 4.2: A Multilayer Perceptron with one input, hidden and output layer [13].

4.8.2 Convolutional Neural Network

The next popular deep learning model is a convolutional neural network (CNN), which mostly used in visual imagery recognition. However, due to their pattern recognition background and ability to capture both temporal and spatial dependencies, they have also been proven to show good performance on multivariate regression data. CNNs are regularized kinds of MLPs. Unlike MLPs, CNNs omit the "full connectivity" property of MLPs, i.e. CNN algorithms penalize the parameters during training or drop out some connections between neurons, thus, avoiding overfitting. CNNs consist of multiple layers: convolutional layer, pooling layer and fully-connected layer.

The former layer uses a specialized type of linear function, called convolution, to detect the low-level patterns from the input data. It applies a sliding filter, or kernel, upon two dimensions of the input to obtain an arbitrary number of feature maps. On the example of image recognition: the convolution operation extracts following features: edges of the image, color or gradient orientation. The more convolutional layers are added to the network, the higher is the ability of the network to adapt to high-level features as well. The pooling layer is responsible for reducing the dimensionality of the feature maps obtained by the convolutional layer. Such operation is denoted as downsampling. Downsampling deals with overfitting and decreases the number of model's parameters, in order to avoid high consumption of computational resources required to process the data. There are two kinds of pooling layers: max pooling and average pooling. Max pooling returns the maximum value from the region covered by the kernel, while average pooling returns the average of all values. A fully-connected, or dense, layer transforms the extracted and downsampled features produced by the two preeceding layers into the final output of the neural network. This layer is followed by an activation function due to the possible presence of non-linearities in the model.

4.8.3 Long Short-Term Memory Network

Long Short-Term Memory Network (LSTM) is a deep learning architecture, which is based on the architecture of recurrent neural networks (RNN). RNNs are distinguished from traditional feedforward neural networks, like CNNS, by their ability to monitor their own temporal dynamic behavior, i.e. they allow previous outputs to be used as inputs for the current node. The state of the network at time t can be described mathematically as follows:

$$h_t = g(Wx_t + Uh_{t-1}),$$

where W is the weight matrix, x_t input of the network, h_{t-1} state of the network at time $(t-1)$ with weight matrix U. However, RNNs are not able to memorize the information from earlier steps if the input sequence is too long. The short-memory problem was solved by LSTMs. LSTM contains feedback connections and is usually composed of a cell, an input gate, an output gate and a forget gate. The key concept of LSTM's is a cell state. It transports the relevant information to the current active node. The gate is a mechanism, which controls the flow of the input sequence by learning which data should be kept or neglected. Gates use sigmoid activation function to normalize the parameters between 0 and 1. The forget gate provides the network with the relevance of the information from prior states. The information from previous state and current input is passed to the sigmoid function. The outputs of the sigmoid function close to 0 will be forgotten and those, close to 1 will be kept in memory. The input gate decides which parameters should be updated based on the information about the current step. This procedure is similar to the one in the forget gate. The next state of the network is derived by the output gate function based on the computations in the prior network gates.

Chapter 5

Online Adaptive Multivariate Time Series Forecasting

Due to the nature of multi-dimensional time series data, it will often contain a lot of irrelevant and redundant information. Hence, it is of paramount importance to be able to determine a subset of the time series for an accurate and efficient forecasting. This task will be further denoted as feature selection, where 'feature' describes one complete time series. The main role of feature selection in time series forecasting is to identify an optimal subset of features, that is capable of describing target concept. Furthermore, as already stated above, non-stationarity is one of the typical characteristics when it comes to time series forecasting problems. Hence, we devise an adaptive learning framework for multivariate time series forecasting. The first aspect of this work is to ensure, that the model is being trained on the right, i.e. the most relevant, subset of features. The second aspect is to constantly monitor changes in the underlying interdependencies (i.e. spatial dimension) as well as in the model's performance (i.e. temporal dimension), as more data is being discovered. The third aspect is adaptivity: the model should be retrained every time there is any change in either spatial or temporal aspects.

The framework operates in an online fashion, i.e. it sequentially processes incoming instances, one at a time, and uses sliding windows on training data for drift detection within both dimensions. This ensures the inclusion of the closest examples to the target one in the computation of similarity between the features and estimated error of the model. The size of the sliding window is a user-defined hyperparameter. The concept drift detection and adaptation ensures, that the validity window of the model trained on the specific input is automatically determined.

Fundamentally, the framework can be split into two components: input feature selection and model performance monitoring. The detailed diagram of the framework is shown below.

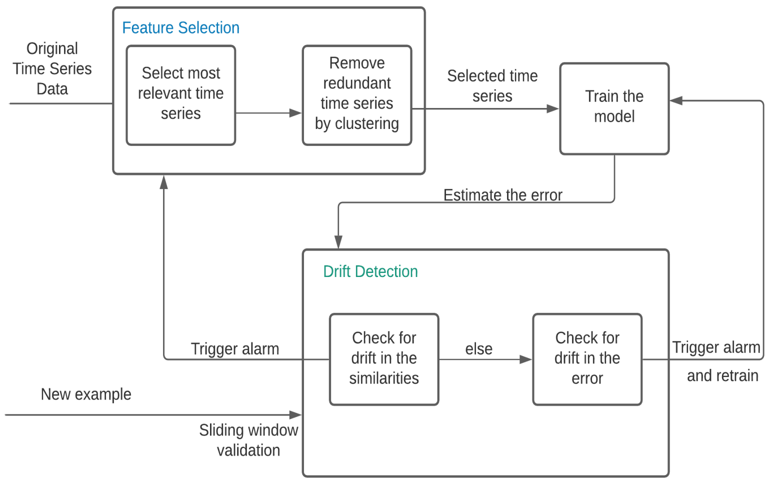
Original Time Series Data
Feature Selection
Select most relevant time series
Remove redundant time series by clustering
Selected time series
Train the model
Estimate the error
Drift Detection
Check for drift in the similarities
else
Check for drift in the error
Trigger alarm
Trigger alarm and retrain
New example
Sliding window validation

Figure 5.1: Feature Selection-Drift Detection Framework.

5.1 Feature Selection

The key goal of feature selection is to find those highly relevant features, which are closely related to the target concept. In this work, we make use of two characteristics of inter-dependent time series: relevance and redundancy. The importance is estimated based on the similarity with the target time series. The task of this step is to examine which of the candidate time series are best suited for the defined forecasting task. Formally, it can be defined as follows:

5.1.1 Definition. (Feature Selection) Find vector $v \in \{0,1\}^M$:

$$v[m] = \begin{cases} 1, & \text{if } F[m] \text{ is relevant and non-redundant} \\ 0, & \text{otherwise} \end{cases}$$

where F is matrix of time series features of spatial dimension $M-1$. Initially, all vector values are assigned to 0.

5.1.1 Similarity

In the first stage, the procedure selects Top-k features most similar to the target one, where k is a user-defined hyperparameter. Due to the varying nature of the time series data, we evaluate different distance and similarity measures presented in Section 2. We normalize the distance-based similarity between 0 and 1 to make the comparison of two measures less complicated.

In fast-evolving environments recent observation are assumed to be more representative for the upcoming situation than old data. For this reason, we constraint the selection to the most recent data, by using the sliding window, which includes the n last instances.

This strategy helps to increase the accuracy of the computed measures while keeping the computational resources low. Hence, the similarity between candidate time series a and target time series b can be described like

$$s(a, b) = g(a_{(t-n+1):t}, b_{(t-n+1):t})$$

where g is a similarity function and t index of the last observed instance. After that, we sort the features in descending order based on their similarity score and select the Top-k most important ones. The vector values of the selected features are set to 1.

5.1.2 Redundancy

In order to ensure diversity within the selected subset of features, we propose to investigate the relation between Top-k selected time series and remove redundancies. This can be achieved by time series clustering. In time series clustering, each feature series consisting of a large amount of data-points can be seen as a single object. As a result, we obtain groups of time series features categorized by their similarity measure and can now select one representative per cluster. In this work, we compared the performance of k-means clustering using two similarity functions: Euclidean and DTW distances.

The main assumption behind it: if two time series are similar in the original space, then their representations should be similar in the embedded space too [2]. The feature vector value of the representatives is assigned to 1, otherwise the assigned value is 0. The cluster representative is the time series closest to the cluster center.

After that, the final selected subset of features and past observations of the target variable are used to train the initial model.

5.2 Drift Detection

The next component of the framework is a Concept Drift Detection unit. It can be divided into two separate steps: drift detection within underlying dependencies in the data and error-based drift detection. In both cases, the model adapts to change by retraining. The term dependencies stands for the inter-dependent relationship between single time series within one dataset.

5.2.1 Drift Detection based on the Dependencies

As we have already mentioned, the distribution of the time series data may follow non-stationary concepts. In this work, we inspect the presence of non-stationarity in the dependencies for a predefined Region of Interest (ROI) for a target variable. ROI is defined by the Top-k random processes $y^i, \forall i \in \mathbb{K}$, selected in the first stage of the procedure. This non-stationarity condition for current ROI is based on the following assumption: if the distance between the two most dissimilar processes within current ROI significantly deviates over time, the drift within dependencies has occurred. This condition can be tested using Hoeffding Bound [15] on the sequentially incoming data using sliding window:

5.2.1 Condition. *Let $M_t^i \in \mathbb{R}^{k \times k}$ be a similarity matrix of a selected ROI, where n is the number of last observed instance. Let ν and ν^t indicate the minimum similarity coefficients at $(t-1)$ and t correspondingly. The drift in underlying dependencies is suspected, if the following condition is violated:*

$$|\nu^{t-1} - \nu^t| \leqslant \sqrt{\frac{\ln(1/\delta)}{2n}}$$

where n is the number of observed instances and δ is a user-defined hyperparameter. The violation of this condition will trigger an alarm in the procedure and force update of the feature vector v computed on the most recent data. Hence, this will cause the retraining of the model using a new selection.

5.2.2 Error-based Drift Detection

One of the most important evaluation metrics, when talking about time series forecasting, is estimated forecasting error. It indicates the difference between the real values and the values induced by the forecasting function f. The increase in the estimated prediction error would signal a possible change in the relationship between the input data and outdated model, due to outdated data, forcing the update of the model's parameters. For this reason, we have employed the well-known drift detection method, namely Page-Hinkley Test. Page-Hinkley Test ensures the detection of drifts in the normal behavior of a stochastic process established by the model [12]. The formal definition of the Page-Hinkley Test was given in 3.3. We present the pseudocode of the Page-Hinkley Test below. ν and ϱ are user-defined hyperparameters, where ν defines the tolerable change in the estimated error and ϱ is a threshold. A larger ϱ value might help to avoid detecting false alarms, but can also miss some drifts [12]. Moreover, the error e is monitored upon a sliding window of fixed size on the recent data. The size of the sliding window has to be adjusted before the procedure. The entailed alert at time t triggers the update of the current model and restarts the Page-Hinkley Test from the beginning.

Algorithm 1 PageHinkleyTest

```
Input: Error at instance $t$: $e_t$; Admissible Change $\nu$; Threshold $\varrho$
Output: Drift at time $t$: $alert_t$
1: /* Initialize CUSUM and the error estimator */
2: $E(0) \leftarrow 0$; $cusum_T \leftarrow 0$; $M_T \leftarrow 1$
3: for $i \in \{1...\infty\}$ do
4:     $alerts_i = 0$
5:     $E(i) \leftarrow E(i-1) + e_t$
6:     $cusum_T(t) \leftarrow cusum(i-1) + e_t - \frac{E(t)}{t} - \nu$
7:     $M_T \leftarrow min(M_T, cusum_T(t))$
8:     if $(cusum_T(t) - M_t) \geqslant \varrho$ then
9:         return $alert_t = 1$
10:    end if
11: end for
```

The steps of the proposed framework are detailed in the pseudocode below. The inputs of the algorithm are initial training subset of time series features F_{train}, target time series t_{train}, t_{test} used for training, parameter k for feature selection; user-defined hyperparameters s similarity function and $clust$ clustering method. Parameters w and e indicate the size of the sliding windows for computation of similarity and estimated error.

In step 2, the initialization of similarity matrix $S^{M \times M}$ and error vector E is done. Steps 6 and 7 are repeated for every candidate time series from the initial training data: the similarity between the target time series t_{train} and candidate feature $f_{train} \in F_{train}$ is computed on the last w observations and stored in the similarity matrix S. Step 9 performs the Top-k selection and clustering using the $clust$ method on the data available for training. $F_{selected}$ is a reduced number of time series features, top_k and $finalSelection$ are the vectors of 0 and 1, defining selected top-K time series and cluster representatives correspondingly. In step 11, the initial model is trained on the selected subset of features and target time series. In step 12, the online time series processing starts. At first, the newly arrived instance is added to the training set, the window is moved to time $(t + 1)$ and similarities are recomputed. In step 19, the algorithm selects $s_i, j \in S$, for which top_{ki} is assigned to 1. This subset is denoted as current ROI. In steps 20 and 21, the minimum similarity measures at time-instances i and $(i - 1)$ are extracted. Step 22 computes the Hoeffding bound on the extracted minimums and, if needed, triggers the alarm. In steps 23-25, the algorithm recomputes the feature selection on the most recent data, given that the alarm was triggered in step 22.

When $(i - e) > 1$, the error drift detection component is activated, where i stands for current time and e for the size of sliding window for error estimation. For each newly observed instance f_i, the algorithm computes the estimated error on the window of size e

using the user-defined error method: RMSE or SMAPE. The error measures are stored in form of a time series of errors. When the error time series has more than two instances available, the algorithm performs the Page-Hinkley test. If the threshold ϱ is reached, the test will output an alert about the possible concept drift and reinitialize the test, if alert was returned.

In steps 34-36, the algorithm checks, if the drift was detected only in the model's performance, in order to update the $F_{selected}$ on the most recent data. The drift in the dependencies would mean, that the $F_{selected}$ has already been updated on the newest data with updated $finalSelection$ vector. Steps 37-39 ensure the update of the model's parameters based on the assumption about the drift in dependencies and estimated performance of the model.

Algorithm 2 DetectAndAdapt

```
Input: $F_{train}; F_{test}; t_{train}; t_{test}; k; s; clust; w; e$
Output: $M$
 1: Initialization:
 2: Initialize similarity matrix and error vector: $S$; $E$
 3: Initialize alerts: $alertDependencies$, $alertError$
 4: Stage 1: Feature selection
 5: for $f_{train} \in F_{train}$ do
 6:   Compute the similarity $s(f_{train}, t_{train})$ on the last $w$ observations:
 7:   $S(1, train) \leftarrow s(f_{train}, t_{train})$
 8: end for
 9: $F_{selected}, top_k, finalSelection \leftarrow$ DoSelection($F_{train}, S, k$)
10: Stage 2: Model initialization
11: $M \leftarrow train(F_{selected}, t_{train})$
12: for (each new instance $f_i$) do
13:   $F_{train}, t_{train} \leftarrow$ Add the newly arrived instance $f_i$ to the training set
14:   Stage 3: Drift detection within dependencies
15:   for $f_{train} \in F_{train}$ do
16:     Compute the similarity $s(f_{train}, t_{train})$ on the last $w$ observations:
17:     $S(i+1, train) \leftarrow s(f_{train}, t_{train})$
18:   end for
19:   Subset Top-$k$ similarities, selected in 9, from $S$: $S_{subset}$
20:   $min1 \leftarrow min(S_{subset}(i-1))$
21:   $min2 \leftarrow min(S_{subset}(i))$
22:   $alertDependecies \leftarrow$ DependenciesCheck($min1, min2$)
23:   if $alertDependecies == 1$ then
24:     $F_{selected}, top_k, finalSelection \leftarrow$ DoSelection($F_{train}, S, k, clust$)
25:   end if
26:   if (i-e)>1 then
27:     Stage 4: Drift detection in model's performance
28:     Compute the training error $err_i$ on the time window $e$
29:     $E(i) \leftarrow err_i$
30:     if $(i-e) \geqslant 2$ then
31:       $alertError \leftarrow$ PageHinkleyTest($E$)
32:     end if
33:   end if
34:   if $(alertError == 1)\&\&(alertDependecies != 1)$ then
35:     Do the old selection $finalSelection$ on the updated dataset $F_{train}$: $F_{selected}$
36:   end if
37:   if $(alertError == 1)\|(alertDependecies == 1)$ then
38:     $M \leftarrow train(F_{selected}, t_{train})$
39:   end if
40: end for
41: return M
```

Algorithm 3 DoSelection

```
Input: Training data $F_{train}$; Similarity matrix $S$; $k$; $clust$
Output: $F_{selected}$; $top_k$; $finalSelection$
1: Select Top-$k$ features from $F_{train}$ with the highest similarity measure from $S$: $F_{kTrain}$, $top_k$
2: Compute the clusters of $F_{kTrain}$ using defined method $clust$ and select cluster representatives: $F_{selected}$, $finalSelection$
3: return $F_{selected}$, $top_k$, $finalSelection$
```

Algorithm 4 DependenciesCheck

```
Input: Min.Similarity at time t $min_t$; Min.Similarity at time (t+1) $min_{t+1}$; Number of observations $W$
Output: Alert at time $t+1$
1: $dif \leftarrow min_t - min_{t+1}$
2: $alert \leftarrow 0$
3: if $|dif| > \sqrt{\frac{\ln(1/\delta)}{2W}}$ then
4:     $alert \leftarrow 1$
5: end if
6: return $alert$
```

Chapter 6

Experiments

In the following chapter we experimentally evaluate our method Drift-Aware Online Time Series Forecasting (DOTSF) and outline the main contributions of the framework by answering the following questions:

1. How good is the performance of DOTSF compared to the classic time series forecasting methods and its own variants?
2. What is the impact of different similarity and clustering strategies across different datasets?
3. Is there any difference in terms of accuracy of proposed approach upon different models?
4. How scalable is the DOTSF compared to the uniformed drift detection methods like periodic retraining, and is there any win in terms of computational resources?

We also compare the proposed method with some variants of itself to evaluate, which part of the concept drift detection component influences the model's accuracy the most.

6.1 Experimental Setup

The time series are preprocessed before the training, using time series embedding. Time series embedding results into a following embedded feature space $X_{t-k}$and the target vector y_k:

$$\mathbb{X}_{t-k} = \begin{pmatrix} y^1_{t-k^1+1} & \cdots & y^1_t \\ \vdots & \vdots & \vdots \\ y^i_{t-k^i+1} & \cdots & y^i_t \\ \vdots & \vdots & \vdots \\ y^M_{t-k^M+1} & \cdots & y^M_t \end{pmatrix} y_k = \begin{pmatrix} y^1_{t+1} \\ \vdots \\ y^i_{t+1} \\ \vdots \\ y^M_{t+1} \end{pmatrix}$$

where k is the size of embedding dimension, M is the spacial dimension of time series and t is the dimension of features. For evaluation we have used 6 real world multivariate time series from various expert domains. The list of all datasets is presented in Appendix A. In each experiment scenario, the time series data was split to 75% for training, and 25% for testing. The embedding dimension of 10 and 5 were applied to different kinds of models.

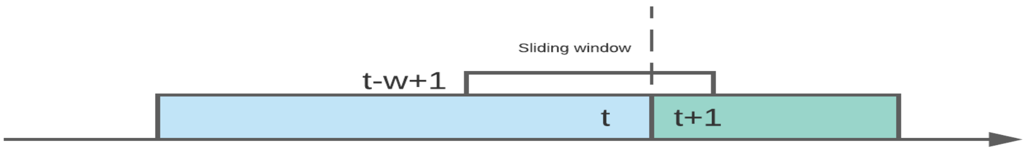

Figure 6.1: One iteration of online data processing. Training data is colored in blue, testing data in green. w is the length of the sliding window.

We calculated RMSE on the predictions of the testing data. Below is the summary of hyperparameters of the framework and the chosen values for the experimental setup:

Parameter	Value
Number of top-k most similar features	25% of the initial number of features
K for k-means clustering	computed using silhouette method
Maximum number of clusters for DTW-clustering	Half of top-k features
Hoeffding bound δ	0.95
Admissible change ν	0.005
Page-Hinkley Test threshold ϱ	0.025
Size of sliding window w	200
Size of sliding window e	20

We evaluate our framework on the following models:

Abbreviation	Description
RF10	Random Forest with embedding dimension 10
RF5	Random Forest with embedding dimension 5
CNN10	Convolutional Neural Network with embedding dimension 10
GBM5	Gradient Boosting Machine with embedding dimension 5
SVM.poly5	Support Vector Machine with polynomial kernel and embedding dimension 5
SVM.laplace5	Support Vector Machine with laplacian kernel and embedding dimension 5
PLS5	Partial Least Squares Regression with embedding dimension 5
PPR10	Projection Pursuit Regression with embedding dimension 10
MARS5	Multivariate Adaptive Regression Splines with embedding dimension 5
MLP5	Multilayer Perceptron with embedding dimension 5
LSTM5	Bidirectional Long Short-Term Memory Network with embedding dimension 5

The performance of our framework is compared against classic machine learning methods used for handling time series data, i.e. offline training and periodic retraining (PR). In offline learning, the model is trained once and tested on the available testing data. In periodic retraining, the model is updated after fixed periodic interval n. n is a hyperparameter, which in our evaluation was set to 25% of the available test data. In order to outline the impact of single components of the proposed framework, we compare our method with some variants of itself:

Offline-Sel-Similarity/Clustering: Uses proposed feature selection to train the model offline.

PR-Sel-Similarity/Clustering: Uses proposed feature selection to retrain the model periodically.

DOTSF-Error-Similarity/Clustering: Performs only error-based drift detection in online time series processing.

DOTSF-Sel-Error-Similarity/Clustering: Performs pre-selection of the features and error-based drift detector on the model, trained on the reduced dataset.

DOTSF-Dependencies-Similarity/Clustering: Uses proposed feature selection and tracks the concept drift only upon the selected dependencies. 'Similarity/Clustering' indicates the corresponding combination of the similarity and clustering measures for each model. The following notation also relates to the DOTSF itself, i.e., 'DOTSF-Similarity/Clustering'.

6.2 Results

In this section we present and discuss the results derived from the experiments listed above. We have tested feature selection using various pairwise combinations of similarity and clustering approaches, including:

Similarity	Clustering
Euclidean distance	k-means clustering
Fourier-based distance	
Manhattan distance	
Pearson correlation	DTW-clustering
Spearman correlation	

Table 6.1: Similarity/Clustering methods.

In order to get a good overview of the test results, we keep three variation of similarity-clustering combination per model family. The first combination is Euclidean-DTW, which is chosen due to its popularity of the measures in the research work and serves as a baseline for the final comparison. The second combination is the trade-off between all the models: the one with lowest RMSE upon all the models and test data, i.e., Manhattan-Kmeans. The third one is chosen based on the lowest RMSE upon all the tests for each model individually. However, if the third choice coincides with either one of the first pairs, then the method with the next lowest RMSE will be chosen.

Tables 6.5 - 6.15 present the paired comparison of the proposed framework against the baseline methods and variants of the framework for each model. We perform the non-parametric Wilcoxon Signed Rank test [34] to compute wins and losses of our method against the methods named above using the RMSE scores. The rightmost column contains the average rank gained by each method and the standard deviation of the rank upon all the datasets. A rank of 1 indicates the best performing model on all multivariate time series. We also calculated the mean of the average rank and the mean of the standard deviation across all the forecasting models. The results are presented in the Table 6.2. "-Similarity/Clustering" stands for the selection method, chosen for each model individually.

Our method has a clear advantage over compared baseline methods in every model family. Both baseline methods, offline and periodic retraining, tend to be more accurate when being trained on a pre-selected subset of time series, using proposed selection method. However, these methods still show a big difference in the average rank compared to DOTSF.

Method	Avg.rank
Offline	8.4±1.1
PR	7.7±0.9
Offline-Sel-Similarity/Clustering	6.7±0.7
DOTSF-Error	6.0±0.7
PR-Sel-Similarity/Clustering	5.9±0.8
DOTSF-Dependencies-Similarity/Clustering	5.2±0.5
DOTSF-Euclidean/DTW	4.4±0.8
DOTSF-Sel-Error-Similarity/Clustering	4.1±0.9
DOTSF-Manhattan/Kmeans	3.6±0.6
DOTSF-Similarity/Clustering	2.9±0.6

Table 6.2: Mean of the average rank and mean of the standard deviation per method, calculated across all the forecasting models.

The two competitive approaches to our method are its own variants: DOTSF-Sel-Error-Similarity/Clustering and DOTSF-Dependencies-Similarity/Clustering. DOTSF-Error shows the worst results between all the variants of the DOTSF. This proves a fact, that time series selection plays a key role in the time series forecasting. Based on the example of the SVM with laplacian kernel (see Table 6.9), we can see a clear advantage of our method over the methods, which use only one of the drift detection components, i.e., either input-based or error-based drift detection. For this example the DOTSF-Spearman/Kmeans has gained 6 out of 6 wins over the DOTSF-Error and DOTSF-Dependencies-Spearman/Kmeans, 4 out of 6 wins over the DOTSF-Dependencies-Spearman/Kmeans. However, DOTSF-Dependencies-Fourier/Kmeans and DOTSF-Sel-Error-Fourier/Kmeans performed significantly better than DOTSF-Fourier/Kmeans when being tested for the PLS with embedding dimension 5 (see Table 6.11).

Yet, there is a clear overall superiority of our method DOTSF-Similarity/Clustering, when using all the proposed components, i.e. time series selection and drift detection within spatial and temporal aspects of the model. Regarding **Question 1**, our results show that DOTSF-Similarity/Clustering evidently outperforms the baselines for time series forecasting and other combinations of itself.

We also compared the particular selection variant for each model individually to two fixed variants upon all the forecasting models. Euclidean similarity measure and DTW-Clustering are considered to be a baseline for time series forecasting. However, the experiments have shown that the averagely best performing similarity/clustering combination tended to be Manhattan-Kmeans. The superiority of the k-Means clustering over DTW-clustering can be explained by the usage of sliding window. Sliding window assures that two time series are similar-sized. Both Euclidean-DTW and Manhattan-Kmeans perform better than customized method twice in each case. The average rank of the Euclidean-DTW is higher than customized and averagely good upon all the models methods. The observations in Table 6.2 answer the **Question 2**, which regards the impact of chosen similarity and clustering metrics for time series selection. There is no universal selection method which can be applied to every kind of data as some of the underlying assumptions(e.g., stationarity) about the time series are not always met. Furthermore, the choice of the sliding window size also plays a crucial role when selecting the right similarity and clustering metrics.

To investigate the performance of the DOTSF-Similarity/Clustering across different models, we summarize the DOTSFs with the best performing selection method in the Table 6.3.

Method	Avg.rank
GBM5	10.8±0.4
SVM.laplace5	9.0±0.9
SVM.poly5	8.8±1.0
LSTM5	8.0±2.2
CNN10	7.5±2.0
RF10	5.5±1.4
RF5	5.0±0.6
MLP5	4.7±1.5
PPR10	3.5±1.4
MARS5	1.8±0.8
PLS5	1.3±0.5

Table 6.3: Mean of the average rank and mean of the standard deviation per model for a fixed selection algorithm DOTSF-Manhattan/Kmeans.

From the table it is obvious, that some models perform **significantly better** than the others. This proves a fact of nonexistence of a model with equally good performance among every kind of time series. This answers the **Question 3** regarding the impact of the chosen forecasting model.

The **Question 4** asks about the impact of informed drift detection on the scalability of the proposed method. This question can be answered by comparing the average runtime of DOTSF-Similarity/Clustering to the average time of uninformed drift detection methods like PR and PR-Sel-Similarity/Clusterng. In the Table 6.4, we see that the average runtime of the periodic retraining is more than 1.5 times greater than the runtime of our method. The runtime has high deviation due to the variety of the models and the datasets of different sizes used for the evaluation.

We also present the RMSE for each model and dataset in Appendix A.

Method	Avg. runtime in sec.
DOTSF-Similarity/Clustering	14.54±29.31
PR	25.49±47.14
PR-Sel-Similarity/Clustering	12.79±26.36

Table 6.4: The runtime comparison between DOTSF and uninformed drift detectors, i.e. PR and PR-Sel.

Method	Our Method		
	Losses	Wins	Avg.rank
Offline	1	5	9.0±1.5
PR	1	5	6.8±2.6
Offline-Sel-Pearson/Kmeans	0	6	8.7±1.7
PR-Sel-Pearson/Kmeans	2	4	6.5±1.4
DOTSF-Error	1	5	4.6±2.1
DOTSF-Sel-Error-Pearson/Kmeans	3	3	3.6±1.9
DOTSF-Dependencies-Pearson/Kmeans	2	4	6.6±2.1
DOTSF-Euclidean/DTW	3	3	**2.3±1.4**
DOTSF-Manhattan/Kmeans	3	3	3.6±2.2
DOTSF-Pearson/Kmeans	-	-	3.4±2.6

Table 6.5: RF10.

Method	Our Method		
	Losses	Wins	Avg.rank
Offline	0	6	9.1±1.4
PR	0	6	6.7±1.0
Offline-Sel-Pearson/Kmeans	0	6	9.4±0.7
PR-Sel-Pearson/Kmeans	0	6	7.4±0.9
DOTSF-Error	1	5	4.8±1.9
DOTSF-Sel-Error-Pearson/Kmeans	3	3	3.6±2.5
DOTSF-Dependencies-Pearson/Kmeans	1	5	6±2.0
DOTSF-Euclidean/DTW	4	2	**2.3±1.0**
DOTSF-Manhattan/Kmeans	4	2	2.8±1.3
DOTSF-Pearson/Kmeans	-	-	3±1.3

Table 6.6: RF5.

Method	Our Method		
	Losses	Wins	Avg.rank
Offline	1	5	6.5±2.4
PR	0	6	8.2 ±2.7
Offline-Sel-Fourier/Kmeans	3	3	3.7±1.8
PR-Sel-Fourier/Kmeans	2	4	4.2±2.7
DOTSF-Error	0	6	5.7±2.5
DOTSF-Sel-Error-Fourier/Kmeans	3	3	3.0±0.7
DOTSF-Dependencies-Fourier/Kmeans	3	3	4.1±1.3
DOTSF-Euclidean/DTW	1	5	7.2±1.9
DOTSF-Manhattan/Kmeans	1	5	7.2±1.9
DOTSF-Fourier/Kmeans	-	-	**2.5±1.5**

Table 6.7: CNN10.

Method	Our Method		
	Losses	Wins	Avg.rank
Offline	1	5	5.8±2.9
PR	1	5	7.3±2.4
Offline-Sel-Fourier/Kmeans	1	5	7.8±2.2
PR-Sel-Fourier/Kmeans	1	5	7±2.5
DOTSF-Error	2	4	4.1±2.2
DOTSF-Sel-Error-Fourier/Kmeans	0	6	5±2.1
DOTSF-Dependencies-Fourier/Kmeans	1	5	8.1±1.6
DOTSF-Euclidean/DTW	3	3	3.5±2
DOTSF-Manhattan/Kmeans	3	3	3.5±3.2
DOTSF-Fourier/Kmeans	-	-	**3±2.4**

Table 6.8: GBM5.

Method	Our Method		
	Losses	Wins	Avg.rank
Offline	0	6	9.5±0.5
PR	0	6	9.5±0.5
Offline-Sel-Spearman/Kmeans	0	6	6.9±1.3
PR-Sel-Spearman/Kmeans	0	6	7.3±0.8
DOTSF-Error	0	6	5.7±1.4
DOTSF-Sel-Error-Spearman/Kmeans	2	4	3.8±2.1
DOTSF-Dependencies-Spearman/Kmeans	0	6	4.6±1.5
DOTSF-Euclidean/DTW	3	3	2.8±1.3
DOTSF-Manhattan/Kmeans	2	4	2.7±1.6
DOTSF-Spearman/Kmeans	-	-	**2.2±1.5**

Table 6.9: SVM.laplace5.

Method	Our Method		
	Losses	Wins	Avg.rank
Offline	1	5	8.3±3.1
PR	1	5	8.3±2.7
Offline-Sel-Manhattan/DTW	0	6	6.4±0.9
PR-Sel-Manhattan/DTW	0	6	7.0±1.1
DOTSF-Error	1	5	6.0±2.8
DOTSF-Sel-Error-Manhattan/DTW	1	5	4.0±2.7
DOTSF-Dependencies-Manhattan/DTW	1	5	5.4±1.7
DOTSF-Euclidean/DTW	2	4	3.0±1.7
DOTSF-Manhattan/Kmeans	2	4	4.0±3.2
DOTSF-Manhattan/DTW	-	-	**2.5±1.4**

Table 6.10: SVM.poly5.

Method	Our Method		
	Losses	Wins	Avg.rank
Offline	0	6	9.8±0.6
PR	0	6	9.0±0.6
Offline-Sel-Fourier/Kmeans	4	2	4.1±0.8
PR-Sel-Fourier/Kmeans	1	5	4.8±1.6
DOTSF-Error	0	6	8.1±0.7
DOTSF-Sel-Error-Fourier/Kmeans	5	1	3.6±0.7
DOTSF-Dependencies-Fourier/Kmeans	5	1	3.6±1.3
DOTSF-Euclidean/DTW	1	5	6.2±2.6
DOTSF-Manhattan/Kmeans	5	1	**2.5±2.4**
DOTSF-Fourier/Kmeans	-	-	3.5±1.7

Table 6.11: PLS5.

Method	Our Method		
	Losses	Wins	Avg.rank
Offline	1	5	8.3±3.4
PR	1	5	7.3±1.8
Offline-Sel-Spearman/Kmeans	0	6	7.5±2.4
PR-Sel-Spearman/Kmeans	0	6	6.5±2.3
DOTSF-Error	1	5	5.7±2.8
DOTSF-Sel-Error-Spearman/Kmeans	3	3	2.8±1.4
DOTSF-Dependencies-Spearman/Kmeans	2	4	5.6±2.3
DOTSF-Euclidean/DTW	1	5	4.8±1.5
DOTSF-Manhattan/Kmeans	1	5	4.0±2.9
DOTSF-Spearman/Kmeans	-	-	**2.4±2.1**

Table 6.12: PPR10.

Method	Our Method		
	Losses	Wins	Avg.rank
Offline	1	5	8.0±1.6
PR	2	4	5.6±2.9
Offline-Sel-Pearson/Kmeans	1	5	8.2±2.3
PR-Sel-Pearson/Kmeans	1	5	5.8±2.5
DOTSF-Error	2	4	5.4±2.4
DOTSF-Sel-Error-Pearson/Kmeans	2	4	5.6±1.0
DOTSF-Dependencies-Pearson/Kmeans	5	1	3.8±3.1
DOTSF-Euclidean/DTW	3	3	4.8±2.2
DOTSF-Manhattan/Kmeans	3	3	4.9±2.1
DOTSF-Pearson/Kmeans	-	-	**3.0±1.6**

Table 6.13: MARS5.

Method	Our Method		
	Losses	Wins	Avg.rank
Offline	0	6	9.5±0.8
PR	0	6	8.5±2.3
Offline-Sel-Euclidean/Kmeans	0	6	5.6±1.4
PR-Sel-Euclidean/Kmeans	2	4	3.4±2.3
DOTSF-Error	0	6	7.2±1.5
DOTSF-Sel-Error-Euclidean/Kmeans	2	4	5.0±3.3
DOTSF-Dependencies-Euclidean/Kmeans	2	4	4.1±2.2
DOTSF-Euclidean/DTW	0	6	6.2±1.6
DOTSF-Manhattan/Kmeans	3	3	3.2±1.9
DOTSF-Euclidean/Kmeans	-	-	**2.2±0.5**

Table 6.14: MLP5.

Method	Our Method		
	Losses	Wins	Avg.rank
Offline	2	4	7.6±3.1
PR	1	5	6.7±1.4
Offline-Sel-Spearman/Kmeans	3	3	4.5±2.9
PR-Sel-Spearman/Kmeans	1	5	5.0±2.6
DOTSF-Error	1	5	8.4±1.5
DOTSF-Sel-Error-Spearman/Kmeans	3	3	4.2±3.2
DOTSF-Dependencies-Spearman/Kmeans	2	4	4.8±1.8
DOTSF-Euclidean/DTW	2	4	6.3±3.6
DOTSF-Manhattan/Kmeans	4	2	**3.3±2.9**
DOTSF-Spearman/Kmeans	-	-	4.2±1.9

Table 6.15: LSTM5.

Chapter 7

Further Discussion

In this thesis, we have presented an online adaptive multivariate time series forecasting framework. Our method performs a dynamic time series selection and updates the model in an informed way based on the drift detection algorithm within temporal and spatial aspects. We have shown that the proposed method was able to outperform the baseline time series forecasting approaches. The two-stage time series selection ensures the inclusion of the most relevant time series and exclusion of the redundant ones by the means of clustering. This generates a low-dimensional training input that includes important and diverse time series to obtain an accurate forecast of the model. We have also demonstrated, that the right combination of similarity and clustering methods plays a crucial role in the time series selection. As a future research, we aim to address the aspect of automatic adjustment to the best similarity/clustering combination, e.g., by the means of meta-learning.

The integrated drift detection component assures the adaption based on the information about the change in the dependencies of input data and the change in the model's estimated performance over time. The motivation behind the drift-aware adaptation mechanism is to update the model and/or selected time series only when the drift is detected. Thus, avoiding the potential exhaustion of computational resources, which is often an issue when employing periodic retraining. Based on the conducted experiments, the periodic retraining needed on average 1.5 more times than our method.

We monitor the drifts using the sliding window approach of the fixed size. However, the approach of monitoring the error can be studied more in future research work by extending it to adaptive windowing. The adaptive windowing solves the following optimization problem: adjust the window size so, that the estimated generalization error on new examples is minimized. This technique may help to reduce the number of false alarms, thus, improving the general accuracy of the model.

The evaluation of the experiments has shown the main advantages of our method over the classic time series forecasting approaches. The time series from different fields were used in the conducted experiments. Further studies could involve a more exhaustive evaluation of the proposed approach on a wider variety of time series.

Appendix A

Appendix

Nr.	Time Series	Data Source	Size	F
D1	Appliances energy prediction data	UCI Machine Learning Repository	2000	29
D2	Daily Bitcoin exchange price	Kaggle	1000	9
D3	Marketing data	Kaggle	104	7
D4	Russel-Stock Market Data	UCI Machine Learning Repository	1985	82
D5	S&P-Stock Market Data	UCI Machine Learning Repository	1985	82
D6	NASDAQ-Stock Market Data	UCI Machine Learning Repository	1985	82

Table A.1: The summary of datasets used for the evaluation.

Method	D1	D2	D3	D4	D5	D6
Offline	0.0033	0.0466	0.0886	0.0727	0.0923	0.1141
PR	0.0032	0.0445	0.0835	0.0430	0.0498	0.0627
Offline-Sel-Euclidean/Kmeans	0.0030	0.0481	0.0820	0.0808	0.1061	0.1138
PR-Sel-Euclidean/Kmeans	0.0030	0.0449	0.0793	0.0488	0.0553	0.0589
DOTSF-Error	0.0032	0.0446	0.0697	0.0368	0.0276	0.0349
DOTSF-Sel-Error-Euclidean/Kmeans	0.0030	0.0452	0.0638	0.0420	**0.0200**	0.0295
DOTSF-Dependencies-Euclidean/Kmeans	0.0028	0.0481	0.0714	0.0527	0.0366	0.0620
DOTSF-Euclidean/DTW	**0.0026**	**0.0431**	0.0637	0.0316	0.0278	0.0305
DOTSF-Manhattan/Kmeans	0.0027	0.0475	**0.0585**	0.0354	0.0281	0.0298
DOTSF-Euclidean/Kmeans	0.0028	0.0475	0.0714	**0.0288**	0.0217	**0.0285**

Table A.2: RMSE for each dataset and method respectively applied to **RF10**.

Method	D1	D2	D3	D4	D5	D6
Offline	0.0030	0.0499	0.0841	0.0707	0.0895	0.1114
PR	0.0028	0.0461	0.0755	0.0441	0.0460	0.0616
Offline-Sel-Euclidean/Kmeans	0.0033	0.0468	0.0805	0.0778	0.1004	0.1092
PR-Sel-Euclidean/Kmeans	0.0031	0.0445	0.0778	0.0457	0.0518	0.0594
DOTSF-Error	0.0030	**0.0426**	0.0727	0.0383	0.0263	0.0315
DOTSF-Sel-Error-Euclidean/Kmeans	0.0031	0.0442	0.0693	0.0289	**0.0188**	**0.0271**
DOTSF-Dependencies-Euclidean/Kmeans	0.0026	0.468	0.0679	0.0506	0.0356	0.0569
DOTSF-Euclidean/DTW	0.0026	0.0429	**0.0620**	0.0286	0.0262	0.0274
DOTSF-Manhattan/Kmeans	**0.0025**	0.0444	0.0653	0.0325	0.0248	0.0273
DOTSF-Euclidean/Kmeans	0.0026	0.0444	0.0666	**0.0272**	0.0214	0.0281

Table A.3: RMSE for each dataset and method respectively applied to **RF5**.

Method	D1	D2	D3	D4	D5	D6
Offline	0.0321	0.1534	0.0931	0.1030	00243	0.0646
PR	0.0486	0.1828	0.1123	0.0892	0.0732	0.0922
Offline-Sel-Euclidean/Kmeans	0.0271	0.0879	**0.0412**	0.0612	0.0233	0.0290
PR-Sel-Euclidean/Kmeans	0.0544	**0.0694**	0.0498	0.0399	0.0210	0.0648
DOTSF-Error	0.0674	0.1345	0.0931	0.0309	0.0243	0.0249
DOTSF-Sel-Error-Euclidean/Kmeans	0.0418	0.0748	0.0412	0.0328	0.0233	0.0249
DOTSF-Dependencies-Euclidean/Kmeans	0.0349	0.0879	0.0412	0.0612	0.0233	0.0290
DOTSF-Euclidean/DTW	0.0749	0.0767	0.0433	0.0696	**0.0198**	0.0722
DOTSF-Manhattan/Kmeans	**0.0268**	0.0746	0.0633	0.0536	0.0305	0.0546
DOTSF-Euclidean/Kmeans	0.0454	0.0697	0.0412	**0.0294**	0.0233	**0.0239**

Table A.4: RMSE for each dataset and method respectively applied to **CNN10**.

Method	D1	D2	D3	D4	D5	D6
Offline	**0.2145**	0.9710	1.1056	0.3321	0.3471	0.3952
PR	0.2164	0.9885	1.0274	0.3332	0.3485	0.4007
Offline-Sel-Euclidean/Kmeans	0.2148	1.0105	0.9148	0.3340	0.3525	0.3956
PR-Sel-Euclidean/Kmeans	0.2164	0.9999	0.9265	0.3323	0.3511	0.3981
DOTSF-Error	0.2226	0.9356	0.8704	0.2415	**0.2063**	0.2389
DOTSF-Sel-Error-Euclidean/Kmeans	0.2227	0.9679	0.8125	0.2417	0.2157	0.2400
DOTSF-Dependencies-Euclidean/Kmeans	0.2207	1.0105	0.9149	0.3340	0.3525	03956
DOTSF-Euclidean/DTW	0.2191	**0.8631**	0.8440	0.2481	0.2179	**0.2326**
DOTSF-Manhattan/Kmeans	0.2246	0.9005	0.7880	0.2399	0.2106	0.2375
DOTSF-Euclidean/Kmeans	0.2226	0.9005	**0.7738**	**0.2396**	0.2121	0.2380

Table A.5: RMSE for each dataset and method respectively applied to **GBM5**.

Method	D1	D2	D3	D4	D5	D6
Offline	0.2053	0.3896	0.9889	0.5492	0.5430	0.5885
PR	0.2047	0.3887	0.9918	0.5515	0.5457	0.5884
Offline-Sel-Euclidean/Kmeans	0.1815	0.2158	0.7528	0.3607	0.4488	0.4767
PR-Sel-Euclidean/Kmeans	0.1959	0.2250	0.7673	0.3565	0.4409	0.4874
DOTSF-Error	0.1352	0.2638	0.6621	0.1299	0.1364	0.1539
DOTSF-Sel-Error-Euclidean/Kmeans	0.1698	0.2361	0.5585	0.0810	0.1060	0.0865
DOTSF-Dependencies-Euclidean/Kmeans	0.0745	0.2158	0.7211	0.0861	0.1586	0.1530
DOTSF-Euclidean/DTW	0.0921	0.2145	0.5764	0.0960	**0.0682**	0.0630
DOTSF-Manhattan/Kmeans	0.1543	0.2117	**0.4796**	0.0822	0.1115	**0.0435**
DOTSF-Euclidean/Kmeans	**0.0513**	**0.1423**	0.5960	**0.0721**	0.1024	0.1085

Table A.6: RMSE for each dataset and method respectively applied to **SVM.laplace5**.

Method	D1	D2	D3	D4	D5	D6
Offline	0.2028	0.1718	1.0206	0.4616	0.4519	0.04931
PR	0.2028	0.1746	1.0154	0.4643	0.4530	0.4796
Offline-Sel-Euclidean/Kmeans	0.1086	0.2060	0.9823	0.0920	0.0814	0.0999
PR-Sel-Euclidean/Kmeans	0.1053	0.2153	0.9784	0.0920	0.0817	0.1000
DOTSF-Error	0.1477	**0.1513**	0.7910	0.0796	0.0980	0.1137
DOTSF-Sel-Error-Euclidean/Kmeans	0.0868	0.2965	0.7577	0.0409	0.0305	0.0423
DOTSF-Dependencies-Euclidean/Kmeans	0.1086	0.2060	0.7120	0.0920	0.0814	0.0999
DOTSF-Euclidean/DTW	**0.0790**	0.2011	0.7303	0.0710	0.0332	**0.0336**
DOTSF-Manhattan/Kmeans	0.0797	0.3316	**0.6188**	0.0529	0.0567	0.0503
DOTSF-Euclidean/Kmeans	0.0884	0.1913	0.7126	**0.0407**	**0.0305**	0.0425

Table A.7: RMSE for each dataset and method respectively applied to **SVM.poly5**.

Method	D1	D2	D3	D4	D5	D6
Offline	0.0032	0.0336	0.0830	0.0475	0.0318	0.0246
PR	0.0035	0.0336	0.0761	0.0226	0.0132	0.0191
Offline-Sel-Euclidean/Kmeans	0.0016	0.0331	0.0639	0.0123	0.0082	0.0105
PR-Sel-Euclidean/Kmeans	0.0016	0.0329	0.0606	0.0123	0.0082	0.0106
DOTSF-Error	0.0032	0.0336	0.0722	0.0186	0.0146	0.0175
DOTSF-Sel-Error-Euclidean/Kmeans	0.0016	0.0331	0.0589	0.0123	0.0082	0.0105
DOTSF-Dependencies-Euclidean/Kmeans	0.0016	0.0331	0.0639	0.0123	0.0082	0.0105
DOTSF-Euclidean/DTW	0.0016	**0.0321**	0.0738	0.0143	0.0098	0.0147
DOTSF-Manhattan/Kmeans	0.0016	0.0331	**0.0535**	**0.0122**	**0.0082**	**0.0104**
DOTSF-Euclidean/Kmeans	**0.0016**	0.0331	0.0589	0.0123	0.0082	0.0105

Table A.8: RMSE for each dataset and method respectively applied to **PLS5**.

Method	D1	D2	D3	D4	D5	D6
Offline	0.0032	0.0420	0.0553	**0.0564**	0.0743	0.0947
PR	0.0027	0.0402	0.0594	0.0338	0.0400	0.0526
Offline-Sel-Euclidean/Kmeans	0.0018	0.0347	0.0672	0.0575	0.0651	0.0943
PR-Sel-Euclidean/Kmeans	0.0019	0.0336	0.1110	0.0305	0.0375	0.0480
DOTSF-Error	0.0032	0.0381	0.0553	0.0256	0.0235	0.0271
DOTSF-Sel-Error-Euclidean/Kmeans	0.0018	0.0329	0.0556	0.0184	**0.0207**	0.0236
DOTSF-Dependencies-Euclidean/Kmeans	0.0016	0.0347	0.0634	0.0427	0.0284	0.0507
DOTSF-Euclidean/DTW	0.0018	0.0376	0.0599	0.0222	0.0202	0.0240
DOTSF-Manhattan/Kmeans	0.0017	0.0349	0.0898	0.0189	0.0152	0.0224
DOTSF-Euclidean/Kmeans	**0.0016**	**0.0329**	0.0634	0.0215	**0.0152**	**0.0213**

Table A.9: RMSE for each dataset and method respectively applied to **PPR10**.

Method	D1	D2	D3	D4	D5	D6
Offline	0.0018	0.0322	0.1369	0.0128	0.0142	0.0149
PR	0.0018	0.0322	0.1035	0.0191	0.0105	0.0110
Offline-Sel-Euclidean/Kmeans	0.0018	0.0323	0.1044	0.0128	0.0143	0.0151
PR-Sel-Euclidean/Kmeans	0.0018	0.0322	0.0994	0.0189	0.0105	0.0110
DOTSF-Error	0.0018	0.0322	0.1001	0.0128	0.0128	0.0135
DOTSF-Sel-Error-Euclidean/Kmeans	0.0018	0.0322	0.0784	0.0128	0.0128	0.0135
DOTSF-Dependencies-Euclidean/Kmeans	0.0018	0.0323	0.0550	0.0126	0.0085	0.0109
DOTSF-Euclidean/DTW	0.0018	0.0322	0.0586	**0.0126**	0.0128	0.0135
DOTSF-Manhattan/Kmeans	0.0018	0.0322	**0.0489**	0.0128	0.0128	0.0135
DOTSF-Euclidean/Kmeans	0.0018	0.0322	0.0550	0.0126	**0.0085**	**0.0109**

Table A.10: RMSE for each dataset and method respectively applied to **MARS5**.

Method	D1	D2	D3	D4	D5	D6
Offline	0.0134	0.0394	0.1759	0.1051	0.1124	0.1027
PR	0.0137	0.0404	0.1204	0.0408	0.0474	0.0429
Offline-Sel-Euclidean/Kmeans	0.0044	0.0361	0.1212	0.0269	0.0466	0.0299
PR-Sel-Euclidean/Kmeans	0.0021	**0.0351**	0.1240	**0.0209**	0.0203	0.0268
DOTSF-Error	0.0092	0.0395	0.1214	0.0321	0.0296	0.0377
DOTSF-Sel-Error-Euclidean/Kmeans	0.0033	0.0364	0.1278	0.0352	**0.0167**	**0.0189**
DOTSF-Dependencies-Euclidean/Kmeans	0.0020	0.0361	0.1154	0.0236	0.0466	0.0299
DOTSF-Euclidean/DTW	0.0031	0.0361	0.1437	0.0297	0.0301	0.0344
DOTSF-Manhattan/Kmeans	0.0034	0.0359	**0.1148**	0.0283	0.0199	0.0221
DOTSF-Euclidean/Kmeans	**0.0020**	0.0359	0.1154	0.0235	0.0171	0.0234

Table A.11: RMSE for each dataset and method respectively applied to **MLP5**.

Method	D1	D2	D3	D4	D5	D6
Offline	1.0000	0.1483	1.0000	1.0000	0.4873	1.0000
PR	0.6270	0.1122	0.2832	1.0000	1.0000	1.0000
Offline-Sel-Euclidean/Kmeans	0.0966	0.0619	0.1004	**0.0354**	1.0000	0.0292
PR-Sel-Euclidean/Kmeans	0.0786	0.8832	**0.0796**	0.6015	0.7934	0.6105
DOTSF-Error	1.0000	0.7173	1.0000	1.0000	0.9750	1.0000
DOTSF-Sel-Error-Euclidean/Kmeans	0.0901	0.0583	0.6775	0.0354	1.0000	**0.0276**
DOTSF-Dependencies-Euclidean/Kmeans	0.0499	0.0619	0.7387	0.6540	0.7929	0.5080
DOTSF-Euclidean/DTW	0.0343	1.0000	0.0871	0.6543	1.0000	1.0000
DOTSF-Manhattan/Kmeans	**0.0302**	**0.0521**	0.6828	0.7483	**0.0229**	0.0303
DOTSF-Euclidean/Kmeans	0.0570	0.7178	0.0887	0.0361	0.6473	0.2695

Table A.12: RMSE for each dataset and method respectively applied to **LSTM5**.

Bibliography

[1] *A comparison of time series similarity measures for classification and change detection of ecosystem dynamics.* Remote Sensing of Environment, 115(12):3129–3152, 2011.

[2] Aghabozorgi, Saeed, Ali Seyed Shirkhorshidi and Teh Ying Wah: *Time-series clustering – A decade review.* Information Systems, 53:16–38, 2015.

[3] Awad, Mariette and Rahul Khanna: *Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers.* Apress, USA, 1st edition, 2015.

[4] Bifet, Albert, Geoff Holmes, Bernhard Pfahringer, Philipp Kranen, Hardy Kremer, Timm Jansen and Thomas Seidl: *MOA: Massive Online Analysis, a Framework for Stream Classification and Clustering.* Volume 11, pages 44–50, 01–03 Sep 2010.

[5] Bifet, Albert, Jesse Read, Bernhard Pfahringer, Geoff Holmes and Indrė Žliobaitė: *CD-MOA: Change Detection Framework for Massive Online Analysis.* 2013.

[6] Breiman, L: *Random Forests.* Machine Learning, 45:5–32, 10 2001.

[7] Breiman, Leo: *Bagging predictors.* Machine Learning, 24:123–140, 2004.

[8] Cerqueira, Vitor, Luís Torgo, Fábio Pinto and Carlos Soares: *Arbitrated Ensemble for Time Series Forecasting*, pages 478–494. 01 2017.

[9] Friedman, Jerome H.: *Multivariate Adaptive Regression Splines.* The Annals of Statistics, 19(1):1–67, 1991.

[10] Friedman, Jerome H.: *Greedy Function Approximation: A Gradient Boosting Machine.* Annals of Statistics, 29:1189–1232, 2000.

[11] Gama, João, Indrundefined Žliobaitundefined, Albert Bifet, Mykola Pechenizkiy and Abdelhamid Bouchachia: *A Survey on Concept Drift Adaptation.* ACM Comput. Surv., 46(4), March 2014.

[12] Gama, João, Pedro Medas, Gladys Castillo and Pedro Rodrigues: *Learning with Drift Detection.* Volume 8, pages 286–295, 09 2004.

[13] Hassan, Abdelazim Negm, Mohamed Zahran and Oliver Saavedra: *Assessment of Artificial Neural Network for Bathymetry Estimation using high Resolution Satellite Imagery in Shallow Lakes: Case Study El Burullus Lake.* International Water Technology Journal, 5, 12 2015.

[14] Heinermann, Justin and Oliver Kramer: *Machine learning ensembles for wind power prediction.* Renewable Energy, 89:671–679, 2016.

[15] Hoeffding, Wassily: *Probability Inequalities for sums of Bounded Random Variables*, pages 409–426. Springer New York, New York, NY, 1994.

[16] J. Gao, W. Fan, J. Han and P. S. Yu: *A General Framework for Mining Concept-Drifting Data Streams with Skewed Distributions.* Proc. of the 7th SIAM Int. Conf. on Data Mining (SDM)., 2007.

[17] Jain, A. K., M. N. Murty and P. J. Flynn: *Data Clustering: A Review.* ACM Comput. Surv., 31(3):264–323, September 1999.

[18] Klinkenberg, Ralf and Thorsten Joachims: *Detecting concept drift with support vector machines.*

[19] Klinkenberg, Ralf and Stefan Rüping: *Concept Drift and the Importance of Examples.* In *Text Mining – Theoretical Aspects and Applications*, pages 55–77. Physica-Verlag, 2002.

[20] Kolter, J. Zico and Marcus A. Maloof: *Dynamic Weighted Majority: An Ensemble Method for Drifting Concepts.* Journal of Machine Learning Research, 8(91):2755–2790, 2007.

[21] Lu, Jie, Anjin Liu, Fan Dong, Feng Gu, João Gama and Guangquan Zhang: *Learning under Concept Drift: A Review.* 2020.

[22] M. G. Kelly, D. J. Hand and N. M. Adams: *The Impact of Changing Populations on Classifier Performance.* Proc. of the 5th ACM SIGKDD Int. Conf. on Knowl. Disc. and Dat. Mining (KDD)., pages 367–371, 1999.

[23] Mierswa, Ingo and Katharina Morik: *Automatic Feature Extraction for Classifying Audio Data.* Machine Learning, 58:127–149, 02 2005.

[24] Moreira-Matias, Luís, João Gama and João Mendes-Moreira: *Concept Neurons - Handling Drift Issues for Real-Time Industrial Data Mining.*

[25] PAGE, E. S.: *Continuous inspection schemes.* Biometrika, 41(1-2):100–115, 06 1954.

[26] RATH, T.M. and R. MANMATHA: *Word image matching using dynamic time warping.* In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 2, pages II–II, 2003.

[27] ROSENBAUM, LARS, ALEXANDER DÖRR, MATTHIAS BAUER, FRANK BOECKLER and ANDREAS ZELL: *Inferring multi-target QSAR models with taxonomy-based multi-task learning.* Journal of cheminformatics, 5:33, 07 2013.

[28] SCHOLZ, MARTIN and RALF KLINKENBERG: *An ensemble classifier for drifting concepts.* 01 2005.

[29] SCHOLZ, MARTIN and RALF KLINKENBERG: *Boosting classifiers for drifting concepts.* Intelligent Data Analysis, 11, 01 2007.

[30] TSAY, R.: *Multivariate Time Series Analysis: With R and Financial Applications.* 2013.

[31] TUNNICLIFFE WILSON, GRANVILLE. *Journal of Time Series Analysis*, 37, 03 2016.

[32] WALD, ABRAHAM: *Sequential Analysis.* John Wiley and Sons, 1st edition, 1947.

[33] WIDMER, GERHARD and M. KUBAT: *Learning in the Presence of Concept Drift and Hidden Contexts.* Machine Learning, 23, 11 1994.

[34] WILCOXON, FRANK: *Individual Comparisons by Ranking Methods.* Biometrics Bulletin, 1(6):80–83, 1945.