FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Assignment of master's thesis

Title:	Explainability in Time Series Classification
Student:	Bc. Narek Vardanjan
Supervisor:	doc. Ing. Kamil Dedecius, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2023/2024

Instructions

A frequent problem in automated time series classification is the ex-post explainability of achieved results. This includes, but is not limited to, the explanation of which features and to which degree influence the final decisions. The aim of the thesis is to perform a theoretical and experimental analysis in this domain. The steps are as follows:

1) Get familiar with time series modeling and classification. Focus on various methods of time series representation and how to use the resulting features for classification.
2) Study the possibilities of ex-post interpretability in classification. Namely, how to determine the discriminant features and their importance in the final decision of the classifier(s).
3) Based on concrete time series, propose a method (methods) for their analyses and explanation. Use freely available datasets, e.g., from http://timeseriesclassification.com.
4) Experimentally implement the method(s) proposed in the previous step. Evaluate and discuss achieved results.

Recommended literature:
[1] R. Mochaourab, A. Venkitaraman, I. Samsten, P. Papapetrou and C. R. Rojas, "Post Hoc Explainability for Time Series Classification: Toward a signal processing perspective," in IEEE Signal Processing Magazine, vol. 39, no. 4, pp. 119-129, July 2022.
[2] C. Molnar, Interpretable machine learning: A guide for making black box models explainable, Available: https://christophm.github.io/interpretable-ml-book/.
[3] E. Štrumbelj and I. Kononenko, "Explaining prediction models and individual

---

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 3 January 2023 in Prague.

predictions with feature contributions", Knowl. Inf. Syst., vol. 41, no. 3, pp. 647-665, Dec. 2014.

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 3 January 2023 in Prague.

Master's thesis

EXPLAINABILITY IN TIME SERIES CLASSIFICATION

Bc. Narek Vardanjan

Faculty of Information Technology
Katedra aplikované matematiky
Supervisor: Ing. Kamil Dedecius, Ph.D.
May 3, 2023

Contents

List of Figures

List of Tables

List of code listings

First and foremost I am deeply grateful to my supervisor doc. Ing. Kamil Dedecius, Ph.D. for his guidance and for introducing me to this field of research. I would also like to thank all the members of CTU FIT, who guided me in my academic path and gave me stable foundations I could leverage in this thesis. Lastly, I would like to express my gratitude towards my grandparents who supported and always believed in my capabilities and the rest of my family who provided me with a stable environment to blossom.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work for non-profit purposes only, in any way that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Praze on May 3, 2023 .....

Abstract

Time series classification is used in a wide variety of real-world problems, from health analysis to food quality assurance. Despite its usage, the extent of understanding the reasons for the classification determined by the model is insufficient. This thesis proposes a novel type of explainable classifier based on ARIMA models, which are heavily used for time series forecasting. The time series dataset is converted to ARIMA coefficients and classified using random forests classifier. The explainability is provided by established explanation methods for tabular data. Besides the ARIMA models, the thesis additionally provides a method for explaining raw time series data, built upon the LIME method.

Keywords time series, ARIMA, explainability, classification, interpretability

Abstrakt

Klasifikace časových řad má rozsáhlé využití v různých oblastech, od zdravotnictví po zajišťování kvality potravin. Navzdory širokému použití však není snadné interpretovat predikční modely. Tato diplomová práce se zaměřuje na návrh nového klasifikačního modelu s jednodušší vysvětlitelností. Model je založen na ARIMA modelech, které jsou často používány pro předpovídání časových řad. Dataset problému je transformován z surových časových řad na hodnoty koeficientů ARIMA modelu a následně klasifikován pomocí haldy rozhodovacích stromů. Vysvětlitelnost je zajištěna pomocí existujících modelů pro klasifikaci tabulkových dat. Kromě navržených ARIMA modelů práce obsahuje také metodu pro vysvětlení chování modelu nad surovými daty časových řad, která vychází z metody LIME.

Klíčová slova časové řady, ARIMA, vysvětlitelnost, klasifikace, interpretabilita

Abbreviations

ADF	Augmented Dickey-Fuller test
AIC	Akaikes information criterion
AR	AutoRegressive model
ARIMA	AutoRegressive Integrated Moving Average model
ARMA	AutoRegressive Moving Average model
BOSS	Bag of SFA symbols
CNN	Convolutional Neural Network
DFT	Discrete Fourier Transform
DTW	Dynamic Time Warping
DWT	Discrete Wavelet Transform
ECG	ElectroCardioGram
EOG	ElectroOculoGram
FFT	Fast Fourier Transform
HIVE-COTE	HIerarchical VotE COllective of Transformation-based Ensembles
ID3	Iterative Dichotomiser 3 - decision trees algorithm
IG	Information Gain
k-NN	k Nearest Neighbors
LIME	Local Interpretable Model-agnostic Explanations
LOFO	Leave One Feature Out
MCB	Multiple Coefficient Binning
NLP	Natural Language Processing
NN	Neural Network
PACF	Partial AutoCorrelation Function
PCA	Principal Component Analysis
RF	Random Forests
RGB	Red Green Blue
ROCKET	RandOm Convolutional KErnel Transform
SARIMAX	Seasonal AutoRegressive Integrated Moving Average with eXogenous factors
SFA	Symbolic Fourier Approximation
SHAP	SHapleys Additive exPlanations
TS	Time Series
UCR	University California Riverside
YPHL	method for generating uniform spherical layers

Introduction

Time series are present in every aspect of human life. They are used to track our health, economy, and political situation over time. It is crucial to understand, how did models arrive at their respective predictions. Explainability is a field of study, that is trying to achieve human interpretable explanations for our predictions. Strategies used by the explainability methods vary. Some use deeper knowledge of the predicting model structure, others do not enforce the rigid structure of the classifier and the explanation is based solely on the output.

The goal of this thesis is to research existing methods for time series explanation and provide new additions to the field. Those newly proposed methods are experimentally implemented and their results are evaluated.

First of all the state-of-the-art literature review is conducted. Different models are examined and tested for their accuracy. The most accurate methods right now, e.g. HIVE-COTE[1], are ensembles of multiple methods. These methods will be excluded from the review, and instead, their building blocks will be described. There are different ways of tackling the issue of classification, notably distance-based methods and feature extraction methods. The difference lies in the preprocessing phase of the dataset. Unlike feature extraction methods, in the case of distance-based methods, the time series data remains effectively unchanged [2]. On the other hand, in feature selection methods, resulting time series are converted to a different representation. An example could be the conversion to a frequency domain using the fast Fourier transform [3], or Catch22 [4]. After the conversion to different domains, tabular classification models, like random forests, can be applied [5]. Shapelets introduce another approach to the problem, by using distinctive parts of time series that are easily explainable [6]. Another well-performing time series classification method is called BOSS and it is a representative of dictionary-based methods[7]. The BOSS method leverages classification methods for natural language processing, by converting the time series into a sequence of words from an alphabet. This ensures a more stable classification of noisy datasets. Lastly, a neural network model called ROCKET is described. It is based on convolutional kernels, which can represent different features of the series, without a conscious effort of their users [8].

For explainability purposes of time series classification, it is feasible to use black box methods, as the research in the field is not that broad. One of the methods can be finding counterfactual explanations on Fourier-derived features [9]. Apart from counterfactuals, the thesis describes additional methods for explainability. The LIME method creates a local linear model, which simulates the explained model [10]. Another approach is based on a game-theoretic concept called Shapley values, which define the solution for a cooperative game. In this case, the goal of the game is to correctly predict the class for a given input. Features that contributed more towards the correct explanation have higher Shapley values. The method then uses these values to establish feature importance [11, 12]. Additionally, more simple ways of establishing feature importance, like permutation feature importance and LOFO methods, are also described in

Chapter 2.

After a thorough understanding of the current literature, the thesis introduces additional methods for classification based on the ARIMA forecasting methods. A newly created dataset containing time series representation based on ARIMA coefficients is funneled to a tabular classificator. The one used after the transformation in this thesis is called random forests. By modeling the classification using ARIMA, the representation can be described as a simple linear combination of time series values. That makes the method highly interpretable. Apart from the methods based on ARIMA, the thesis also introduces an explainability method using LIME with a custom approximation function tailored for time series data. The function is derived from the version used for image classification. These methods are described theoretically; then implemented in Python, using conventional interfaces from the scikit-learn library [13].

Ultimately, after the description, the methods proposed in this thesis are tested on UCR database [14]. First, the accuracies and prediction speed of the proposed methods are compared to current best-performing models. Then explanations are visualized to exhibit their advantage in interpreting the model behaviors. This thesis aims to explore a challenging and significant field, while the primary objective is to discover an innovative solution. Even if the methods are lacking in performance or explainability to their counterparts, they will still be valuable in exploring various approaches and ideas.

Chapter 1

Classification of time series

The chapter describes what time series is and then the state-of-the-art for the classification of time series datasets.

To understand this thesis, there are essential prerequisites that need to be familiar to the readers. They should understand supervised learning and classification, especially of tabular data. Being comfortable with time series modeling and forecasting is also essential. This chapter provides a necessary introduction to the time series analysis and classification.

1.1 Time series

Time series is a sequence of data points that appear in successive order over some time. Both continuous and discrete time series exist, but this thesis deals only with time series data that are discrete. More generally, it is possible to view the time series as a sequence of ordered observations. Time series modeling is applied to many classification problems that may not intuitively appear as time series problems. For example, the BeetleFly dataset is used to differentiate between beetle and flies using their outlines plotted as a 1-D time series [15]. Example data points are seen in Figure 1.1. The classification methods here are concerned only with the univariate time series, which do not contain multiple values for each index. Further generalization to multi-variate time series classification is not part of this thesis.

▶ **Definition 1.1** (Time series)**.** *Let $(\Omega, \mathcal{F}, P)$ be a probability space and $T \subset Z$ be a set of indices. Then $\{X_t \mid t \in T\}$ is called time series, where X_t is a random variable from the probability space $(\Omega, \mathcal{F}, P)$.*

There are two main ways of representing time series: in the time domain, as defined above, or in the frequency domain. The conversion between the time and frequency domains can be accomplished using Fourier transform and its inverse [3]. Furthermore, wavelets can be used as a compromise between the two methods for converting time series into time-frequency representation.

By staying in the time domain, it is possible to model the time series as a joint probability distribution of random variables X_t from the probability space.

▶ **Definition 1.2** (Joint probability distribution for the time series)**.** *Let X_t be a time series. Then $p_{X_1,\dots,X_n}(x_1, \dots, x_n) = \mathrm{P}(X_1 = x_1 \text{ and } \dots \text{ and } X_n = x_n)$ is the joint distribution of the X_t, where x_t is the observed value of the random variable X_t.*

The time series can be described by their moments, as with any other random variables. The most relevant moments are mean and variance. Additionally, auto-covariance represents the

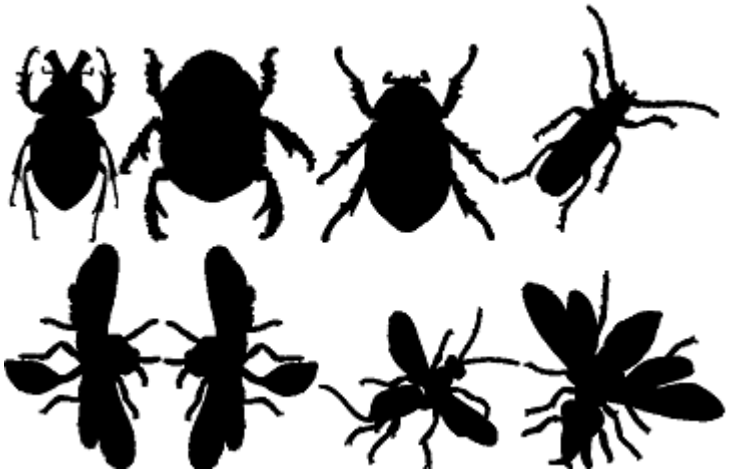


Figure 1.1 Example of a data set that is not intuitively solvable using time series classification models.

covariance between the two different time points of the same time series X_t. The variance can be defined as $\text{var}X_t = \text{cov}(X_t, X_t)$ and is closely related to the auto-correlation.

Mean $\mu_t = \mathbb{E}[X_t]$

Variance $\sigma_t^2 = \text{var}X_t$

Auto-covariance $\text{cov}(X_{t_1}, X_{t_2}) = \mathbb{E}[(X_{t_1} - \mu_t)(X_{t_2} - \mu_t)]$

It is possible to decompose the series into components, which add up to the original data points [16]. Seasonal decomposition for example decomposes a series into seasonality, trend and noise. An example of the time series decomposition can be seen in Figure 1.2.

A long-term increase or decrease in the data is contained in the trend component. Seasonal patterns embody recurrent events and their effects on the time series data, like the end of the week or year. The residuals (error component) of the time series represent the unexplained part of the decomposition. After the decomposition, the residuals should be equal to the white noise, which means no more information from the residuals could be extracted [17].

▶ **Definition 1.3** (Additive and multiplicative decomposition)**.** *Let X_t, T_t, S_t, E_t be time series, then the additive or multiplicative decomposition of the time series is defined as*

$$X_t = T_t + S_t + E_t$$

or

$$X_t = T_t \cdot S_t \cdot E_t,$$

where T_t, S_t, E_t are the trend, seasonality, and error parts of the series respectively. The transformation between the additive and multiplicative seasonal decompositions is possible using the product logarithmic identity:

$$log_c(ab) = log_c(a) + log_c(b).$$

Having stationary data point in the time series is crucial for time series modeling using some models. Stationary data is hard to forecast because there are no predictable patterns in the long term. Formally, it means that the joint distribution of the time series data points is not affected by shifting the series by some time duration. It can be hard to accomplish strong stationarity so weak stationarity is more prevalent as it does not affect the solution significantly [19]. Data can be made stationary using time series differencing. The process is going to be described in later chapters.

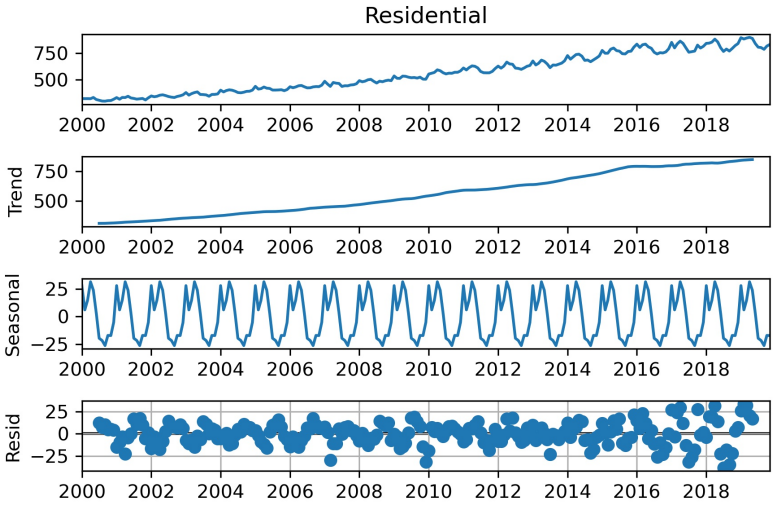


Figure 1.2 A seasonal decompose of an artificial time series [18].

▶ **Definition 1.4** (Strong stationarity)**.** *Let X_t be a time series, then the time series is strongly stationary iff for every $t_1, ..., t_k \in T$ and some duration τ the following holds*

$$p_{X_1,...,X_k}(x_1, \ldots, x_k) = p_{X_\tau,...,X_{k+\tau}}(x_\tau, \ldots, x_k + \tau).$$

▶ **Definition 1.5** (Weak stationarity)**.** *Let X_t be a time series, then the time series is weakly stationary iff for each τ time duration given equations holds*

$$\mathbb{E}[X_t] = \mathbb{E}[X_{t+\tau}],$$

$$\text{cov}(X_t, X_{t+\tau}) = \gamma(\tau).$$

1.2 Time series classification

The problem of time series classification consists of assigning labels from a set Y to a set of series $\mathbf{X}_t$. There is a finite number of classification labels for which the model is trained. Training is primarily done in a supervised manner, giving the model data to learn from. It could also be helpful to group time series data into classes that are not predetermined. That is called unsupervised classification or clustering.

▶ **Definition 1.6** (Classification model)**.** *Let $\mathbf{X}_t$ be a set of time series and Y be a set of labels to classify the time series into, then the mapping function $f : \mathbf{X}_t \to Y$ is called a classification model.*

There are multiple strategies used to build classification models. The common approaches involve feature extraction or distance measuring. Feature extraction methods transform $\mathbf{X}_t$ to a different dataset. By doing so, the dataset becomes easier to classify, because it better represents important patterns, the noise is filtered out and has a lower dimensionality overall.

The distance-measuring approach tackles the problem from a different angle. It compares different time series and classifies the new observation based on its neighbors. The neighborhood is determined using a similarity metric and can be represented as a distance between the corresponding points of the time series. The choice of the similarity metric plays a vital role in classification accuracy. Such functions commonly used are the Euclidean distance and dynamic time-warping [14].

The thesis researches representative samples of different methods. It does not cover some methods used in Chapter 5 because they are ensembles of different classification algorithms. Members of ensembles are described instead in this chapter individually.

1.3 Distance-based classification using k-NN

K-Nearest Neighbors, or in short k-NN, is an example of a distance-based method. It works independently of the similarity-measuring function. This gives the method flexibility to design the function specifically for the domain. Its variant 1-NN is widely used in the time series classification context. In order to use 1-NN, the computation of distance to all other neighbors in the set is needed. It is crucial to have a time series similarity metric that is independent of the phase. If shifts in time series lead to significantly different comparison results, the distance metric is defined as phase dependent.

▶ **Definition 1.7** (Euclidean distance for time series pair)**.** *Let X_t, Y_t be time series, then the mapping D:$(X_t, Y_t) \to \mathbb{R}$ defined as*

$$D(X_t, Y_t) = \sum_{i=1}^{T} \sqrt{(x_i - y_i)^2}$$

is called one dimensional Euclidean distance of time series X_t, Y_t [20].

This distance-measuring function is sadly phase-dependent and only works with a time series of equal sizes. For these reasons, dynamic time warping or DTW was invented, which allows for shifting of the compared data points [21].

▶ **Definition 1.8** (Dynamic time warping)**.** *Let $X_t, Y_u, t \in T, u \in U$ be time series and let $A^{t,u}$ define a matrix, where column indices define ordered random variables from X_t and row indices define ordered random variables from Y_t. Then the element $A^{i,j}$ contains an Euclidean distance between the random variable value X_i and X_j. Dynamic time warping tries to find a path π that minimizes*

$$DTW_q(X, Y) = \min_{\pi \in (T \times U)} \left(\sum_{(i,j) \in \pi} \sqrt{(x_i - y_i)^2} \right).$$

Dynamic time warping relaxes rules for mapping the data points. In contrast, while the Euclidean distance enforces strict rules, the warped paths generated by dynamic time-warping need to conform only to the following conditions:

- $(i_0, j_0) = (0, 0)$
- $(i_k, j_k) = (t, u)$
- $i_{k-1} \leq i_k \leq i_{k-1} + 1$
- $j_{k-1} \leq j_k \leq j_{k-1} + 1$

The first two rules state, that the first and last data points should be mapped to their respective counterparts. The other two are requiring that the sequence should be monotonically increasing for both i and j indices and every index should be used at least once. Combined, they ensure that every data point is assigned and the assignment is correct, which means there is no point in the series where the hypothetical assignment lines cross. This fact can be seen in Figure 1.3.

The version presented in Algorithm 1 works with time series of the same length, but this is an artificial limitation. Additionally, for both performance and correctness reasons, it is advised

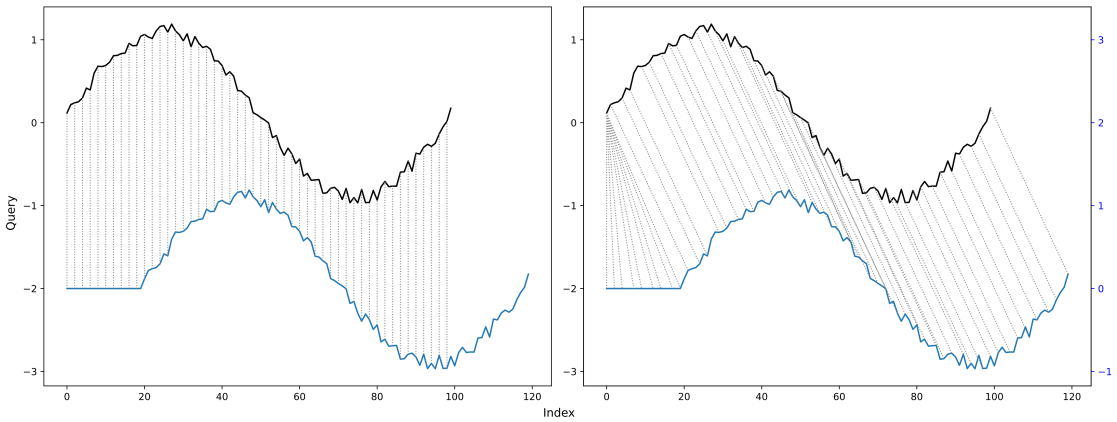


Figure 1.3 Example illustration of the difference between the DTW and the Euclidean distance. It is possible to see the warping of the comparison points in the DTW case.

to not compare every possible combination of time series indices. For that, there is a version of the DTW with locality constraints to penalize comparing far indices [22].

The DTW algorithm involves iterating through all the indices of two sequences and computing the minimum distance by choosing between three cases at each step:

- Skip X_i,
- Skip Y_j,
- Match X_i to X_j.

Then it returns the final minimum value as the result of DTW mapping.

Algorithm 1 Computation of the DTW distance

Require: $X_t, Y_t \in \mathbf{X}_t$ time series
1: $d \leftarrow$ matrix of the size $t \cdot t$ with values equal to ∞
2: $d[0,0] \leftarrow 0$
3: **for** $i \leftarrow 1..t$ **do**
4: **for** $j \leftarrow 1..t$ **do**
5: cost $\leftarrow \text{dist}(X_i, Y_j)$
6: min $\leftarrow \text{minimum}(d[i-1,j], d[i,j-1], d[i-1,j-1])$
7: $d[i,j] \leftarrow \text{min} + \text{cost}$
8: **end for**
9: **end for**
10: **return** $d[t,t]$

The described method is an example of dynamic programming, which means the the solution is computed from the bottom up non-recursively.

1.4 Feature extraction-based methods

As said in the introduction to this chapter feature extraction methods transform the dataset into a more palatable format for the already known classification models. It is important to describe the state-of-the-art models used for the classification of tabular data. Prominent ones include the random forests and gradient boosting algorithms, which are examples of ensemble learning

classifiers. An ensemble is a group of smaller models that together participate in the prediction of the result. Each of them can be additionally weighted to control their influence on the overall classification.

▶ **Definition 1.9** (Tabular classification)**.** *Let $\mathbf{X}^{n,p}$ be a tabular dataset of n data points and X^p be an ordered set of p features and Y a set of labels. Then, the mapping $f : X^p \to Y$ is the classification model for tabular data.*

Both gradient boosting and random forest models provide their users with the mapping f, which can be used for the classification. There is a predecessor to these models called the decision trees model, which is often a building block for ensemble learning algorithms.

1.4.1 Classification using decision trees

A decision tree is a fundamental model used for classification. It traverses the tree and on each node chooses the next child based on the decision criteria. Decision criteria are binary questions related to the features of the observation. This process is repeated until a leaf node is reached. Every leaf node contains a classification label. The classification assigned to an observation is determined by the label of the leaf that the observation reaches.

The selection of appropriate splitting criteria is crucial in decision tree algorithms, as it directly affects the accuracy and performance of the resulting model. Moreover, it is important to generate shallow trees that have high prediction power because deep trees can cause overfitting. One strategy for building a decision tree is called ID3 [23]. The algorithm picks questions that minimize the entropy. The decision is based on local entropy only, which means that it is an example of a greedy algorithm. Greedy algorithms are making decisions based on the local information, for example neighboring nodes in the graph. These can produce suboptimal solutions because they do not consider the global picture of the problem.

▶ **Definition 1.10** (Entropy)**.** *Let Y be a set of classes, S be a dataset containing classes Y and $p : Y \to \mathbb{R}$ be the proportion for the given class $y \in Y$ in the dataset S. Then the entropy can be computed using*

$$H(S) = -\sum_{y \in Y} p(y) \log_2 p(y).$$

Zero entropy means that the S contains only one class from X. It can be further used to compute the information gain.

▶ **Definition 1.11** (Information gain)**.** *Let S be a set of classes from Y and $T \subseteq S$ is a subset of S, then $IG : (S \times T) \to \mathbb{R}$, defined as*

$$IG(S,T) = H(S) - \sum_{t \in T} p(t) H(t),$$

is the information gain achieved by the subsetting of S.

These two definitions are the sole requirements for understanding the ID3 algorithm. In each step, this greedy algorithm picks a feature from the dataset, which splits the classes in a way that maximizes the IG. Chosen question is then used as the inner node and the same algorithm is applied recursively to the newly created children. The process continues until the single class leaves are reached. The process of classifying a newly arrived feature set can be achieved through tree traversal.

1.4.2 Classification using random forests

Random forests are an ensemble learning method, that builds hundreds of decision trees and then takes into account their predictions to create the final one [5]. This strategy is feasible both for classification and regression problems. For the classification, every tree predicts the class on its own. The final result is then equal to the majority of votes. Those can be further weighted if some decision trees are more effective in predicting the correct class.

▶ **Definition 1.12** (Classification using the random forest)**.** *Let X be the input variable from the dataset $\mathbf{X}$, Y be the set of classification labels, and $f_b \in B, f_b : X \to \mathbb{R}$ be a function representing a decision tree from the set of trees B. Then, function $f : X \to Y$ defined as*

$$f(x) = \frac{1}{|B|} \sum_{f_b \in B} f_b(x)$$

is an ensemble of decision trees called the random forest and the output of the function is computed as the average of probabilities for all the classes from Y.

Ordinary decision trees are used as a base for random forests. The build process of the decision trees is explained in Algorithm 2. This is the simplest version of the model. More profound versions not only sample the feature values but also incorporate the omission of certain features in the analysis process.

Algorithm 2 Building random forest classifier

Require: $\mathbf{X}$ time series dataset
Require: Y set of time series classifications for $\mathbf{X}$
Require: n hyperparameter for the number of trees
1: $B[n]$ - array of learners
2: **for** $b = 1..n$ **do**
3: $\quad \mathbf{X}_b, y_b \leftarrow \text{sample}(\mathbf{X}, Y)$
4: $\quad B[b] \leftarrow \text{ID3}(\mathbf{X}_b, y_b)$
5: **end for**
6: **return** B

The advantage of the random forest model is that its variance is decreased with only a slight increase in the bias.

1.4.3 Classification using gradient boosting

An alternative ensemble learning algorithm to random forests is called gradient boosting. It creates weak learners iteratively, each improving the prediction of its predecessor. Every weak learner additively contributes to the final classification. When classifying they add up to the probabilities of labels [24].

▶ **Definition 1.13** (Weak learner)**.** *Let $\mathbf{X}$ be a tabular dataset containing $X \in \mathbf{X}$ data rows and $Y, |Y| = c$ be a set of classes. Then, $F_m : X \to \mathbb{R}^c$ is a weak learner which adds probability to every class in the class set.*

Having the weak learner F_m it is possible to construct the F_{m+1} weak learner by training it on the residuals between the y_i and $F_m(x_i)$. That means the next weak learner is trained only on part of the classification probability, that was not already explained by the previous learners.

▶ **Definition 1.14** (Gradient boosting)**.** *Let X be a tabular dataset with $x_1, ..., x_n$, $y \in Y$ be a set of classification labels, and F_m is a weak learner. Then the improved weak learner F_{m+1} is created using*

$$F_{m+1} = y - F_m.$$

The gradient boosting algorithm can be generalized to a gradient descend algorithm if the loss function used for the training is differentiable. In the case of the classification problem, one could use a categorical cross-entropy.

▶ **Definition 1.15** (Categorical cross-entropy)**.** *Let Y be the set of classifications $|Y| = c$, $x \in \mathbf{X}$ the data point from dataset, and $f : X \to Y$ the model to evaluate. Then*

$$L(Y, \mathbf{f}(x)) = -\sum_{j=1}^{c} \mathbb{1}_{Y=j} \log f_j(x)$$

is the categorical cross-entropy loss function, where $\mathbf{f}(x) = (f_1(x), ..., f_c(x))$ and

$$\mathbb{1}_{Y=c} = \begin{cases} 1 & \text{if } Y = c. \\ 0 & \text{else.} \end{cases}$$

The pseudo-code for random forests is described in Algorithm 3. It is based on the implementation from Friedman [24]. In classification problems using gradient boosting, the cross-entropy function is commonly utilized as the loss function, and the ID3 algorithm with a fixed tree size is typically employed as the weak learner.

Algorithm 3 Creation of the gradient boosting ensemble

Require: $\mathbf{X}$ dataset with N data points
Require: Y classification for data points in $\mathbf{X}$
Require: M the number of weak learners/iterations
Ensure: $F_M(x)$ gradient boosting model
1: $a \leftarrow \frac{1}{|Y|} \sum_{y \in Y} y$
2: $F_0 \leftarrow a$
3: **for** $i \leftarrow 1...M$ **do**
4: **for** $j \leftarrow 0...N$ **do**
5: $r_{i,j} \leftarrow -\left(\frac{\partial L(Y, \mathbf{F_{m-1}}(x))}{\partial \mathbf{F_{m-1}}(x)} \right)$
6: **end for**
7: $h_m = ID3(\mathbf{X}, r_i)$
8: $\gamma_m = \text{argmin}_\gamma \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$
9: $F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$
10: **end for**
11: **return** $F_M(x)$

1.4.4 Time series classification in the frequency domain

While it is feasible to classify time series data using methods typically employed for tabular data, there are several challenges associated with this approach. One such issue arises when dealing with large time series datasets, as the dimensions of the dataset can increase substantially, leading to computational challenges and increased risk of overfitting. Additionally, it can be difficult for models to extrapolate patterns from the raw time series data. The raw time series transformation from the time to frequency domain can solve this problem.

Fourier transform is a way of transforming the time series data to the frequency domain. There exists an additional transformation that can persist the position of the given frequency in the series called a Wavelet transform. Both these transformations are going to be explained in the following chapters with an example usage for time series classification.

1.4.4.1 Fourier transform

The Fourier transform named after Joseph Fourier is an algorithm for the decomposition of any function into sums of sine and cosine functions [3]. As this thesis focuses on discrete time series classification, the more general continuous version of the Fourier transform is going to be left out.

▶ **Definition 1.16** (Fourier transform). *Let X_t be a time series with t data points. Then the following formula*

$$\hat{X}_k = \sum_{j=0}^{n-1} X_j e^{-i2\pi kj/n}$$

is used to get the Fourier coefficient $\hat{X}_k \in \mathbb{C}$

Coefficients extracted from the Fourier transform can be used as features in the new dataset. For that reason, this method is from the feature extraction family of classifiers. It is possible to use the coefficients in their raw form or additionally transform them. A lot of the time it is inevitable because the classification methods do not work well or at all with complex numbers.

One of the simplest ways to convert the coefficients to real values is by taking the absolute value of the complex number. This process is not lossless and some information about the series is lost. Another way to transform the complex number is to create a dataset containing imaginary and real parts separately. This makes $2n$ number of features. A paper from Mochaourab et al. [9] uses different features derived from Fourier coefficients. The values they use are the amplitudes and angles of the complex numbers.

▶ **Definition 1.17** (Computing FT amplitudes). *Let $\hat{X}_t$ be the coefficients of a time series X_t of a length t, then an amplitude A_t can be computed using the following formula*

$$A_t = \frac{2}{t} \cdot abs(\hat{X}_t),$$

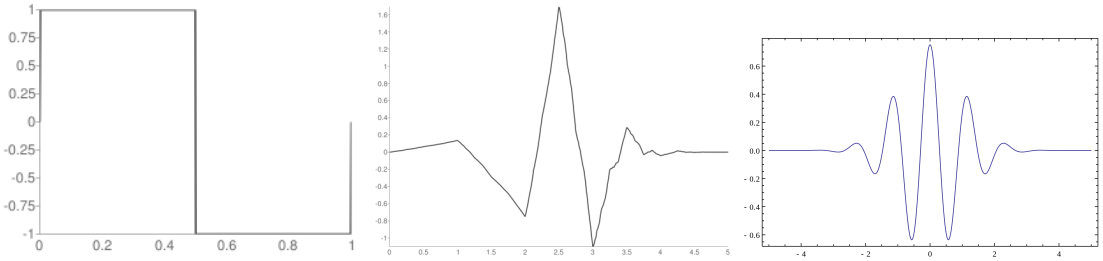
where abs is the absolute value function.

The operation does not change the number of features needed as is apparent from Definition 1.16. To reduce the dimensionality of the dataset, coefficients can be sorted and then only those passing a threshold are left. Sorting is done based on the amplitude value and the threshold is set to filter out the lower amplitudes. Higher amplitudes are related to higher coefficient values.

The discrete Fourier transform is an algorithm that computes the Fourier coefficients, but this version of the algorithm is slow. It takes $O(n^2)$ time to compute the coefficients. For long time series, it is then vital to make the algorithm more time effective. The algorithm called Fast Fourier Transform was invented for this purpose. It achieves the same result using only $O(n \log n)$ steps. There are different versions of the algorithm. The most notable one is called the Cooley-Turkey algorithm [3].

1.4.4.2 Wavelet transform

The Fourier transform does not preserve information about where a particular frequency happens, so it is possible to lose important information about the locality. The wavelet transform was made to fix that problem. It does not just show us frequency but also when those frequencies happen. That information can be crucial for some classification and time modeling problems. For that reason, an additional transformation called wavelet transform was created. It does not convert the time series into the frequency domain. Instead, after applying the wavelet transform, the resulting time series is in the time-frequency domain [25]. These work better in situations where the time series is non-stationary. For a stationary time series, frequency components are relevant over their whole run.



(a) Haar wavelet. **(b)** DB3 wavelet. **(c)** Morlet wavelet.

Figure 1.4 Examples of mother wavelets.

The building block of the wavelet transform is called the wavelet. The Wavelet transform is characterized by its ability to be flexible with the choice of wavelets employed in the transformation process. This attribute sets it apart from the Fourier transform. Many different transforms exist that extract diverse information from the time series. The wavelet, transformed using linear operations, is often called the mother or analyzing wavelet. The linear operations are scaling using a scalar and the addition of wavelets. Figure 1.4 shows examples of some mother wavelets.

Lets now define the continuous wavelet transform and then focus on its special case of the discrete wavelet transform.

▶ **Definition 1.18** (Continuous wavelet transform)**.** *Let g be a function of time and $\psi_{\tau,s}$ be the prototype wavelet function, where τ is the time localization, and s is the scale defined as $1/f$ and f is a frequency. Then, the function F is the continuous wavelet transform of the function g defined as*

$$F(\tau, s) = \frac{1}{\sqrt{|s|}} \int_{-\infty}^{+\infty} g(t)\psi^* \left(\frac{t-\tau}{s} \right) dt.$$

The function above can be used to generate wavelet coefficients, but if τ and s are continuous, there will be vast amounts of coefficients (theoretically infinity). Those parameters are often discretized to prevent that from happening by taking only the powers of two. To compute for example, the coefficient C_{jk}, this equation is used

$$C_{jk} = F(2^{-j}, k2^{-j}).$$

As mentioned in Chapter 1.4.4.1, discrete time series are the sole focus of this thesis. Therefore, it is necessary also to define wavelet transformation with discrete time. This is a particular case of continuous transformation.

▶ **Definition 1.19** (Discrete wavelet transform)**.** *Let X_t be a time series, $\psi_{a,b}$ be the prototype wavelet function where a be the time localization, s be the scale defined as $1/f$, and f is the frequency. Then, the function D is the discrete wavelet transform defined as*

$$D(a, b) = \frac{1}{\sqrt{b}} \sum_{t=0}^{n-1} f(t)\psi^* \left(\frac{t-a}{b} \right).$$

where ψ^* is the complex conjugate of the basis function ψ. Former invariants of a, b being dyadic and resulting coefficients being computed as

$$C_{jk} = D(2^{-j}, k2^{-j}).$$

The ϕ function, which is the mentioned mother wavelet, needs to be chosen for classification purposes. Mochaourab et al. [9] try two wavelet functions, namely the Daubechies and Haar

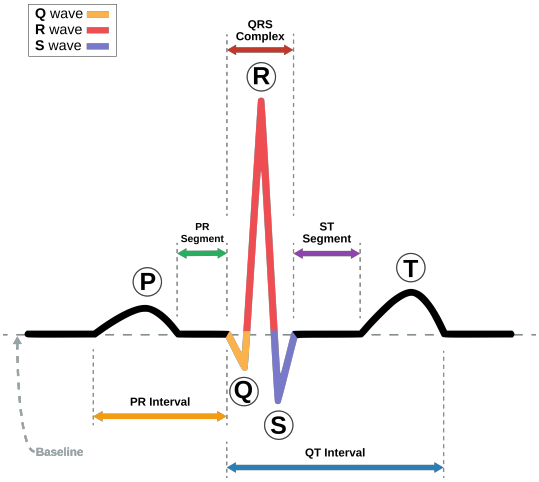


Figure 1.5 Different intervals in the ECG signal [26].

transforms. The Daubechies transformation is useful to model signals like ECG, with a very predictable form and non-smooth form. The one they used is called Db3. The Haar transform is, on the other hand, one of the first wavelets developed and is equal to the Daubechies transform called Db1 [9].

1.4.4.3 Classification on the coefficients

After feature extraction from the time series data, it is time to use some classification models on the newly created dataset. The dataset consists of k numerical features after the transformation. As mentioned in Chapters 1.4.3 and 1.4.2, ensemble methods exist for classifying these datasets called random forests and gradient boosting.

1.4.5 Interval-based classification methods

Interval-based classification methods decompose given time series into smaller intervals, which are then used to train weak learners. They do not have access to the rest of the intervals, so it is harder for the learner to overfit the data. Creating an ensemble of these learners makes it possible to classify the time series. Sub-intervals created in the splitting phase can be mined for mean, standard deviation, slope, and range of other features. The features are then used to train a weak learner.

The value of the method lies in classifying time series data, where only some parts are essential to classify the time series correctly. For example, given an ECG signal, there are different intervals, as seen in Figure 1.5. The thesis introduces an interval-based method called Time series forest.

The random forest approach of Deng et al. [27] uses a random forest classifier for the resulting model containing computed moments of the interval. The number of intervals created equals $\sqrt{m}$, where the m is the time series length. The number of resulting features generated by partitioning a given dataset into $\sqrt{m}$ intervals is $3\sqrt{m}$. The classification outcome is determined based on a voting scheme among the ensemble of weak learners, with the majority decision being taken as the final prediction.

The building block for the random forest approach is the time series tree. It is an example of a decision tree but with different splitting criteria. Original criteria for splitting trees, as seen in the ID3 algorithm, use entropy or information gain. Time series forests use a combination of entropy and distance-based metric.

▶ **Definition 1.20** (Threshold splitting). *Let f_k be a feature computing function, where*

- $f_1(t_1, t_2) = mean(X_{t_1..t_2})$
- $f_2(t_1, t_2) = std(X_{t_1..t_2})$
- $f_3(t_1, t_2) = slope(X_{t_1..t_2})$

Then the condition

$$f_k(t_1, t_2) \leq \tau,$$

where τ is a numerical threshold, is used in a node to forward the time series to one of its children.

The threshold τ candidates are fixed for each feature function f_k. From the list of thresholds, the best one is chosen by its ability to divide the intervals into-equal width bins [27]. Lastly, it is vital to select the correct split. The paper proposes their novel splitting criterion called the entrance.

▶ **Definition 1.21** (Entrance criterion). *Let IG be the information gain between the parent and their child and f_k^n is the result of f_k for the n-th instance at the node, then entrance criterion is defined as*

$$E = IG + \alpha M,$$

where M is defined as

$$M = \min_{n=1,..,N} |f_k^n(t_1, t_2) - \tau|.$$

When the resulting value E is equal to zero for all the different feature functions, splits, and intervals, the node is declared as a leaf, and its label is equivalent to the label of its contained time series. Algorithm 4 describes the entire tree-building process.

Algorithm 4 Building time series tree

Require: X_t data
T $\leftarrow$ sampled $\sqrt{t}$ intervals
Thresh$_k$ $\leftarrow$ calculated set of candidate thresholds
$E' \leftarrow 0, \Delta\text{Entropy}' = 0, t_1' = 0, t_2' = 0, \tau' = 0, f' = 0$
for $t_1, t_2 \in$ **T do**
    for $k \leftarrow 1..K$ **do**
        for $\tau \in$ Thresh$_k$ **do**
            calculate ΔEntropy and E for $f_k(t_1, t_2) \leq \tau$
            if $E \geq E'$ **then**
                $E' \leftarrow E, \Delta\text{Entropy}' = \Delta\text{Entropy}, t_1' = t_1, t_2' = t_2, \tau' = \tau, f' = f_k$
            end if
        end for
    end for
end for
if $\Delta\text{Entropy}' = 0$ **then**
    set node as leaf
    return node
end if
node$_l$ $\leftarrow$ time series with $f'(t_1', t_2') \leq \tau$
node$_r$ $\leftarrow$ time series with $f'(t_1', t_2') \geq \tau$
return node

A time series forest is a collection of time series trees. It leverages the random sampling strategy from the [5], where the features of the observation intervals are sampled instead of

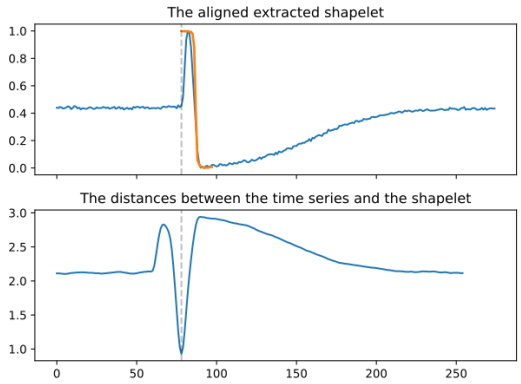


Figure 1.6 Alignment between the extracted shapelet (orange series) and the time series. [28, 29]

sampling the tabular dataset. For each node with a X_t time series start and length of the interval is sampled both $\sqrt{N}$ times. This creates N different samples in each node. By having hundreds of time series trees of this type, the model's accuracy is comparable to random forests. Mainly its ability to not overfit.

1.4.6 Shapelet-based methods

Shapelet-based methods try finding patterns in the time series that sufficiently differentiate classes [6]. These are phase independent and could occur anywhere. This method is robust for time series, where a part is representative of the whole. The question is which part of the series should be used as shapelets. It is possible to craft the shapelets manually by leveraging domain knowledge, but generating them is much more convenient. One of the ways to classify the time series data is to try to find patterns within the series that could occur anywhere. Those patterns are called shapelets in the time series jargon and are phase independent.

There are two distinct methods for generating shapelets. The first is called the shapelet-extracting method, which extracts essential shapelets from the training data. The second one takes a different approach and learns the shapelets from data. It is important to note that the learned shapelets may not correspond precisely to any of the subsequences present in the training data. This method is called the shapelet-learning algorithm.

The time series is compared to the shapelet using sliding window segments. These are equal to the length of the shapelet.

▶ **Definition 1.22** (Sliding window segment)**.** *Let X_t be a time series of the length t, then a new time series $X_{j,l}$ derived from the original X_t as*

$$X_{j,l} = (X_j, ..., X_{j+l-1})$$

is called the sliding window segment of the X_t time series.

1.4.6.1 Shapelet-extraction method

The original version of the method introduced by Ye et al. [6] is an example of a shapelet-extracting method. It uses a brute-force algorithm to arrive at optimal shapelets. To leverage the method, it needs two ingredients. Those are the shapelet and the corresponding optimal split point.

Before defining the optimal split point, some additional definitions need to be provided. Firstly, shapelets need to be comparable to the time series. A distance function $d(X_t, S_u)$ needs to be defined for that.

▶ **Definition 1.23** (Shapelet distance). *Let X_t, $S_u, u < t$ be time series with a presupposition that S_u is a shapelet, then*

$$d(X_t, S_u) = \min(dist(X_{j,u}, S_u) \mid j \in [1, t-u])$$

is the shapelet distance where $X_{j,u}$ is the sliding window version of the X_t over all its segments.

▶ **Definition 1.24** (Distance threshold). *Let $\mathbf{X}_t$ be a set of time series with corresponding labels Y, and IG be the information gain function. Then $d^* \in \mathbb{R}$ is the distance threshold, which splits the dataset $\mathbf{X}_t$ into $\mathbf{X}_t^r$, $\mathbf{X}_t^l$ so that*

$$d(\mathbf{X}_t^l, S_u) < d^*,$$

and

$$d(\mathbf{X}_t^r, S_u) \geq d^*.$$

Having these definitions in place, it is possible to proceed with defining the optimal split point.

▶ **Definition 1.25** (Optimal split point). *Let $\mathbf{X}_t$ be a set of time series with corresponding labels Y, IG be the information gain function, S_u be a subsequence of some series from $\mathbf{X}_t$, $d(X_t, S_u)$ be the distance function of the minimal distance between the time series and shapelet. Then the optimal split point is such distance threshold d_{res} that satisfies*

$$IG(\mathbf{X}_t, S_u, d_{res}^*) \geq IG(\mathbf{X}_t, S_u, d^*)$$

for any other distance threshold d^*, where IG computes the information gain for the newly created sets using the given distance threshold.

Optimal split points say that the given threshold is the best possible because no more information can be gained using any other threshold. The second question is how the shapelets should be generated. In this version of the method, they are extracted using a brute force algorithm seen in Algorithm 5.

Algorithm 5 Finding the best shapelet

Require: $\mathbf{X}_t$ data
Require: *max_len* maximum shapelet length
Require: *min_len* minimum shapelet length
candidates ← *generate_candidates*($\mathbf{X}_t$, *max_len*, *min_len*)
best_gain ← 0
best_shapelet ← *null*
*best_d** ← *null*
for $S_u \in$ candidates **do**
    $gain, d^* \leftarrow IG(\mathbf{X}_t, S_u)$
    if $gain \geq best\_gain$ **then**
        best_gain ← *gain*
        best_shapelet ← S_u
        *best_d** ← d^*
    end if
end for
return *best_shapelet*, *best_d**

The procedure for finding the best shapelet is used to build a decision tree. On each node, a shapelet is created from the remaining training data, and a threshold is used to distinguish how should the new observation continue the traversal. As is apparent from the algorithm, it could be more efficient. The algorithm's complexity is cubic in the number of time series and quadratic in the length of the shapelet [6]. This problem is solvable using the second approach, which is called the shapelet-learning algorithm.

1.4.6.2 Shapelet-learning algorithm

The shapelet-learning tries to learn the shapelets from the dataset. The best shapelet could be missing from the existing data, which would cause the extraction method to be suboptimal. Additionally, as outlined in the preceding chapter, the extraction version has high complexity. The version of the shapelet-learning described in this section is based on the work of Grabocka et al. [30].

Firstly, the shapelet distance defined in 1.23 is modified to return the minimal distance for the given segment. A segment is defined using the starting point in the time series and the length of the shapelet.

▶ **Definition 1.26** (Distance for j segment)**.** *Let $X_t, S_u, u < t$ be time series and $d(X_t, S_u)$ be the shapelet distance. Then $d_j(X_t, S_u)$ is the distance for jth segment defined as*

$$d_j(X_t, S_u) = \frac{1}{u}\sum_{l=1}^{u}(T_{j+l-1} - S_l)^2.$$

The minimum function is then defined as $M = min_{j=1,\dots,J} d_j(X_t, S_u)$, where J is the total number of segments. It is not differentiable, which is a prerequisite for methods based on gradient learning. Replacing the exact minimization with an approximation using the soft-minimum function is possible.

▶ **Definition 1.27** (Soft-Minimum version of the shapelet distance)**.** *Let $d_j(X_t, S_u)$ be the shapelet distance for segment j. Then, its soft minimum approximation M' is defined as*

$$M \approx M' = \frac{\sum_{j=1}^{J} d_j(X_t, S_u) e^{\alpha d_j(X_t, S_u)}}{\sum_{j=1}^{J} e^{\alpha d_j(X_t, S_u)}},$$

which is equal to the true minimum function as $\alpha \to \infty$.

It is apparent from this definition that the shapelet is the single variable in this equation per time series. Leveraging this information, learning the shapelets using gradient descent is possible.

1.4.7 Dictionary-based methods

Both shapelet and interval-based methods presuppose that distinct features in the time series data can accurately classify its category. However, there are situations where the distinction does not lay in the concrete pattern but in the context in which it occurs and the prevalence in given time series data. Additionally, both shapelets and interval-based methods operate directly on the raw data, making them susceptible to noise in the dataset. It is vital to remove the noise by human preprocessing before the application.

A family of dictionary-based methods tackles these issues by sliding over the time series with a sliding window and creating words from the given window and alphabet Σ. As the state-of-the-art contender for this class of methods, BOSS was chosen [7].

1.4.7.1 Bag of SFA symbols (BOSS)

The BOSS model is a 3 step method. Firstly, the time series is converted to a symbolic representation using the symbolic Fourier approximation [31]. SFA is computed on the time series sliding windows, where two consecutive windows share $l-1$ positions. These windows are z-score normalized to achieve consistent scale and vertical alignment.

▶ **Definition 1.28** (Z-score normalization)**.** *Let X_t be a time series, μ be the mean of the series, and σ be the variance of the series. Then, X'_t is the z-score normalized series defined as*

$$X'_t = \frac{X_t - \mu}{\sigma}$$

.

Symbolic Fourier approximation is a dimensionality reduction algorithm based on mapping Fourier coefficients to a discrete set of values, which can be interpreted as symbols. By symbolizing the time series, it is possible to utilize algorithms working with characters or strings (such as a bag of words or hashing). An example of numerical dimensionality reduction is the DFT, where only the first k coefficients are considered.

The algorithm applies either an equi-depth or equi-width binning technique to each occurrence, which refers to the instances where the variable appears in the dataset. Equi-depth binning divides the variable into bins of equal frequency, whereas equi-width binning divides the variable into bins of equal width.

▶ **Definition 1.29** (Histogram)**.** *Let $\mathbf{X}_t$ be a time series dataset of a size n approximated with w Fourier coefficients. Then the value of $H_i(x)$ represents the total number of approximations for which the ith coefficient t_i is equal to x.*

Having the histograms, discretization of them can be achieved by binning using MCB.

▶ **Definition 1.30** (MCB equi-width binning)**.** *Having the approximation for N time series using the FT and a symbol alphabet $\Sigma = s_0, ..., s_c$, the MCB determines wedges $\omega_i(0) \leq ... \leq \omega(c)$ such that two subsequent wedges*

$$\langle B_i(a-1), B_i(a))$$

have equal interval widths. Consequent wedges define discretization intervals.

The discretisation interval can be defined as $DI_i(a) = \langle B_i(a-1), B_i(a))$. Using this notation, the SFA transformation can be defined as follows.

▶ **Definition 1.31** (SFA Word)**.** *Let X_t be a time series, then after discretisation $DFT_w(X_t) = t_1, ..., t_w$ to the first w coefficients of the Fourier transform, it is possible to create word $s = s_0, ..., s_w$ from the alphabet Σ by this definition*

$$s_i \equiv symbol_a \iff t_i \in DI_i(a)$$

The BOSS model creates these SFA words for the sliding window of a length $w \in \mathbf{N}$. The sliding windows are normalized to have a standard deviation equal to 1 and optionally the mean to 0. This step is followed by transforming each sliding window into the SFA word, which creates an unordered set of them.

When encountering a stable section in the time series, BOSS will repeat the exact words multiple times. To avoid giving more power to the stable sections of the time series, numerosity reduction is applied. If the word is repeated successively, only the first occurrence is added to the bag. If a word appears multiple times, it can still be added to the bag of words each time it appears, as long as there is at least one different word between each occurrence.

▶ **Example 1.32** (Example of numerousity reduction)**.** Given the words from the sliding window $S = aab\ aab\ aab\ bcc\ aab$, then the result after numerosity reduction is equal to $S' = aab\ bcc\ aab$.

The final operation on the data is converting these words to their occurrence histograms $B : \Sigma^l \to \mathbb{N}$. This is the last prerequisite before the classification, which is made by leveraging the nearest-neighbors algorithm using the BOSS distance.

▶ **Definition 1.33** (BOSS distance)**.** *Let B_1, B_2 be two BOSS histograms, then the BOSS distance is computed using following formula*

$$dist(B_1, B_2) = \sum_{a \in B_1, B_1(a) > 0} (B_1(a) - B_2(a))^2.$$

1.4.8 Catch22 features

What if there can be a set of features that can be extracted from any time series and are distinctly able to help in the classification? Catch22 tries to find exactly these kinds of features[4].

The creators of the method were able to arrive at 22 distinct features that still preserve a high level of classification and, at the same time, contain a minimum amount of redundancy [4]. They were able to select these through highly comparative time series analysis of freely available datasets.

The feature set contains different features such as distribution parameters, autocorrelation, and the delta between successive differentiations. Some examples of catch22 features are listed below.

DN_HistogramMode_5 Mode of the z-score distribution estimated using 5-bin histogram.

FC_LocalSimple_mean3_stderr Mean error of the 3-sample mean forecasting.

FC_LocalSimple_mean1_tauresrat Change in the correlation length after iterative differencing.

DN_OutlierInclude_p_001_mdrmd Time-intervals between really unlikely ($p < 0.001$) events above the mean.

The selection process was based on 93 freely available datasets [15], which are also used in this thesis. They made performance feature selection to quickly filter features that have the accuracy of a null-distributed classifier.

After the performance analysis, the feature set was further trimmed by ranking the features based on their accuracy normalization. Then they created a cluster of similarly performing features with high correlation (± 0.8), and one representative of the cluster was chosen based on the following:

Normalized accuracy This was computed again across the tasks.

Manual picking That was based on the interpretability of the feature, which is also important.

1.5 Classification using CNN and Convolutional Kernels

Neural networks are state-of-the-art methods for image classification. Time series being effectively an image with one less dimension suggests they should also be effective for classifying time series datasets. Most of the methods perform feature selection by representing the data set rigidly. For example, shapelets are trying to find shapes that are representative of the classification, BOSS is trying to convert the time series to NLP representation and FFT methods use the coefficients to transform the representation to lower dimension.

Convolutional neural networks try a different approach by letting the model learn the representation that gives the most predicting power. They do not approach the problem with preconceived representation but rather generate random convolutional kernels to detect the patterns in the series. This has its downsides, mostly in the interpretability department. Because the kernels are random and do not have concrete interpretation, this model is effectively a black box. One of the representatives is called ROCKET [8] and will be described in this thesis.

▶ **Definition 1.34** (Convolution kernel for time series)**.** *Let X_t be a time series and K_u time series where $u < t$. The equation*

$$s(t) = \sum_{a=-\infty}^{\infty} X_a \cdot K_{t-a}$$

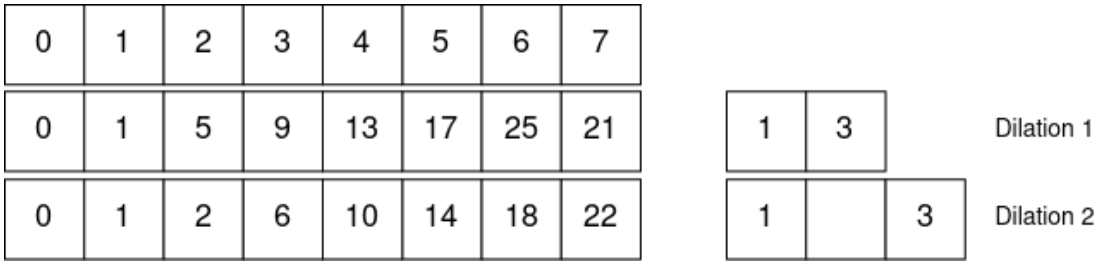


Figure 1.7 Convolution operator application with dilation 1 and 2.

often written $(X * K)(t)$ *is the convolution operation and the time series* K_u *is the convolution kernel. The result of this operation is often called a feature map.*

These convolutional kernels can capture many time series features used in other methods. They can find patterns using appropriate kernels the same way shapelets do. The frequency information can be extracted with kernel dilation. This technique creates gaps in the kernel where the time series value is not considered when running the convolution operation. The dilated convolution can be seen in Figure 1.7. Larger dilation corresponds to lower frequencies, and higher dilations to higher ones [8].

The ROCKET method defines only one layer of convolutional kernels, and the weights of the kernels are not learned, which means that the features are fast to compute. Additionally, because there is only one layer, generating a large number of kernels is possible. Because the kernels are generated randomly, the only hyperparameter of the ROCKET method is the number of kernels. The attributes of kernels that are generated randomly are as follows

Length is selected randomly from 7, 9, 11 with equal probability

Weights are not learned but sampled from the normal distribution and mean centered after being set

Bias is sampled from a uniform distribution and added to the kernel output

Dilation is used to extract frequencies and scales

Padding on each kernel generation, an equal probability decision is made whether or not padding will be used when applying the kernel.

A stride of a kernel is always equal to 1. There could be a worry that these settings were overfitted on the UCR datasets, but the original paper randomly selected half of the UCR dataset to set these parameters and then tested if the classifier could generalize on the rest of the datasets not included in the development ones.

After the application of a kernel, ROCKET computes two aggregate features from each feature map, producing two real-valued numbers per kernel. That means for a k number of kernels, the resulting dataset contains $2k$ features. For smaller datasets, this results in much higher dimensions than the original dimensions for the time series.

Maximum value is equivalent to the global max pooling used in image classifications.

Proportion of positive values defines the input proportion matching a given pattern. It is a novel feature of the method developers.

When the method is finished transforming the dataset, a linear classifier is used on the kernels to compute the weights for the logistic or ridge regression. The ridge regression was chosen in the original paper because it performed better on smaller datasets. Although these are common in the UCR database, it may be beneficial to consider using logistic regression instead when dealing with larger datasets. Additionally, it is not that important to regularize for larger datasets.

Chapter 2

Explainability for time series classification

The content of this chapter is concerned with the explanation methods available in the field of time series classification. It introduces readers to both black-box and white-box strategies of the explanation.

Over the years, time series explainability methods have emerged to help interpret the classifications predicted by the models. Some methods are specific to time series, while others leverage the long-running development in tabular data classification algorithms.

The purpose of this chapter is to summarize existing developments and have an overall understanding of the field. There are different ways to categorize explainability methods. One such categorization is concerned with whether the method explains the model as a whole or if it explains only one classification. This distinction is called the global/local distinction. Another way of categorizing explainability methods is to classify them based on the amount of information needed to do the explainability. White-box models are characterized by their ability to work with the internal representation of the model, unlike black-box models, which rely solely on the model's output.

2.1 Permutation feature importance

Permutation feature importance is a technique used to determine the importance of features in a given dataset for a predictive model. This method belongs to the family of feature importance models and is an example of a black-box model, meaning it can be used with any machine learning model.

It works by permuting feature values between the data points to generate new modified dataset, on which a metric is computed and compared to the same metric on the original dataset[5]. A concrete metric for the comparison is not important to the model, and any accuracy scoring metric can be used.

By doing this, the new dataset rows are more likely to be valid than when the values in rows are picked randomly. The described algorithm has problems when features in the dataset are highly correlated. If that is the case, it can lead to falsely important features because more essential features are not detected by being effectively present multiple times.

One way of solving that could be removing highly correlated columns by applying some feature selection algorithm. The principal component analysis (PCA)[32] could be used, but because the resulting features are harder to interpret, its applicability in explainable models

Algorithm 6 Computation of the permutation feature importance

Require: m fitted model
Require: T tabular dataset
1: $s \leftarrow$ reference score for T
2: **for** feature j from column of T **do**
3: **for** $k \leftarrow 1..K$ **do**
4: $\hat{T}_{k,j} \leftarrow$ D with shuffled feature j
5: $s_{k,j} \leftarrow$ computed score for $\hat{T}_{k,j}$
6: **end for**
7: $i_j \leftarrow$ for feature j as $s - \frac{1}{K}\sum_{k=1}^{K} s_{k,j}$
8: **end for**
9: **return** i importance scores for every feature

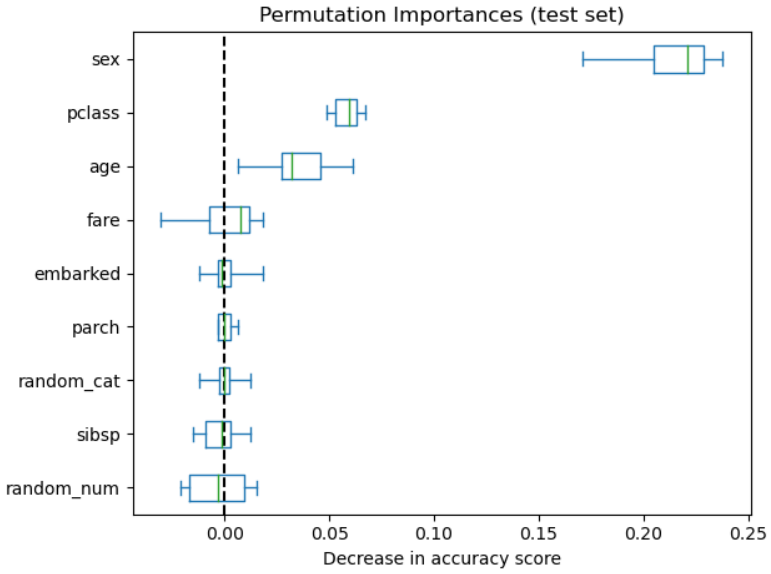


Figure 2.1 Feature importance of the survivors from the Titanic dataset [33, 13]. It tells that by permuting the titanic dataset sex feature between different data points, the decrease in accuracy score is the highest.

is limited. Another more reasonable approach could be clustering the features based on the correlation and considering only one feature from it.

Figure 2.1 shows an example method rendering. The differences in importance are visualized using the whiskers plot to show the uncertainty of the importance. In this type of visualization, it is possible to see additional information about the mean and the outliers on both sides. The observation from the current graph is that sex was more important than any other feature because women had priority when deciding who was going to end up in lifeboats.

2.2 Counterfactual explanations

Unlike the feature importance family of methods, counterfactual explanations do not tell which features are the most important. Instead, they give us the minimum change needed to change the classification from one category to another. These methods are local, and there are both black-box and white-box versions.

▶ **Definition 2.1** (Counterfactual). *Let $m : T \to Y$ be a fitted model, t be a feature set for which $y = m(t)$, y' the desired classification, and $d : T \times T \to \mathbb{R}$ function to measure the distance between the feature sets. Then z' for which the following holds*

$$\min_{t' \in T} d(t', t) \text{ while } y' = h(t')$$

is called a counterfactual.

There are many ways to generate counterfactuals. Mochaourab et al. used an algorithm called Growing Spheres[34] in their post-hoc explainability paper.

The original paper defines the concept of ally and enemy, where *ally* is the original row in the dataset with classification y, and the enemy is the modification of the ally that gives a different classification result. Even though nowhere in the paper is the counterfactual mentioned, this is the exact definition of one.

▶ **Definition 2.2** (Growing spheres distance (cost function)). *Let e, x be the enemy and ally rows, then the distance $c : X \times X \to \mathbb{R}^+$ is defined as*

$$c(x, e) = ||x - e_2||_2 + \gamma ||x - e_0||_0,$$

where $||.||_2$ is the Euclidean norm and $||.||_0$ the l_0 norm, which is defined as the number of non-zero coordinates, $||e - x||_0 = \sum_{i \leq d} 1_{x_i \neq e_i}$ and γ is the penalization hyperparameter.

The reason for the l_0 norm is that humans are imperfect at interpreting multiple features changing simultaneously, so it is preferable to change the least number of features possible (preferably one). This function is discontinuous, so it is tough to find the solution exactly. Therefore, the paper proposes the growing sphere heuristic to get an approximate solution. The approximation of this algorithm is called the growing spheres. It consists of two phases called the generation and feature selection phases.

2.2.1 Generation

In this phase, a greedy approach is used to find the enemy by generating instances in spherical distances from the ally observation. To be more precise, the algorithm defined in the 7 generates l_2-spherical layers around the ally observation until the enemy is found.

▶ **Definition 2.3** ((a_0, a_1)-spherical layer). *Let a_0, a_1 be two positive numbers, then*

$$SL(x, a_0, a_1) = z \in \mathbf{X} : a_0 \leq ||x - z||_2 \leq a_1,$$

defines the spherical layer around the observation x.

An algorithm called YPHL [35] generates these spherical layers of observations uniformly over the sphere. At the start, the a_0 is set to zero and the a_1 is set to hyper-parameter η. Apart from this hyper-parameter, the algorithm defines the number of generated observations in each radius n.

If the algorithm does not find an enemy in the given radius, it shifts a_0, a_1 values and repeats. After the solution is found, it is time to make the enemy as sparse as possible. That is done in the next step.

2.2.2 Feature selection

The feature selection is a heuristic algorithm with a simple strategy. It tries to create sparser enemies by gradually zeroing minor parameters. The idea is that smaller parameters are less likely to step outside of the decision boundary. The implementation is outlined in Algorithm 8.

Algorithm 7 Growing spheres generation

Require: $f : \mathbf{X} \to Y$ classifier
Require: ally $\in \mathbf{X}$
Require: τ, n hyper-parameters
e $\leftarrow SL(x, 0, \tau)$
while enemy $\in$ **e do**
    $\tau \leftarrow \frac{\tau}{2}$
    e $\leftarrow SL(x, 0, \tau)$
end while
$a_0 \leftarrow \tau$
$a_1 \leftarrow 2\tau$
while enemy $\notin$ **e do**
    e $\leftarrow SL(x, a_0, a_1)$
    $a_0 \leftarrow a_1$
    $a_1 \leftarrow a_1 1\tau$
end while
return $e \in$ **e** nearest from the set

Algorithm 8 Growing spheres feature selection

Require: $f : \mathbf{X} \to Y$ classifier
Require: $x \in \mathbf{X}$
Require: $e \in \mathbf{X}$ where $f(e) \neq f(x)$
$e' \leftarrow e$
while $f(e') \neq f(x)$ **do**
    $e^* \leftarrow e'$
    $i = \arg\min |e'_j - x_j|$
    $e'_i \leftarrow x_i$
end while
return e^*

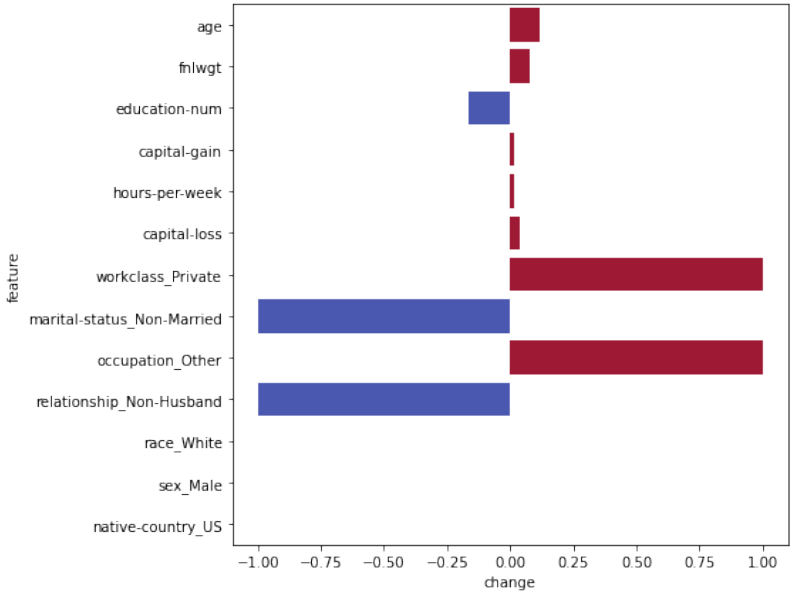


Figure 2.2 Example of a counterfactual from the adult dataset [36].

After the feature selection resulting enemy can be visualized, for example, as a difference between the enemy and ally.

Figure 2.2 shows the difference between the ally and enemy counterfactuals. It could also be interpreted as what should change to trigger the change of the classification outcome. The data are from the adult dataset, where the problem is about predicting if the given people's salary is higher or lower than $50k.

2.3 Shapley values

Shapley values are a game-theoretic approach to explainability, where each feature value is a player, and the prediction is the payout of the game [37]. Shapley values tell us how much the payout should be given to each feature value, effectively telling the predicting power of the given value.

The algorithm first generates every possible combination of features (coalitions) and computes its resulting value. Then feature x we are interested in gets introduced into the coalition to see how it changes the result. Marginal contribution is computed as the difference between the feature sets with and without a given feature. The resulting contribution is then the average overall contributions computed for different coalitions.

As one can deduce, this method has a problem because it is computationally expensive to get the resulting value for every combination of the coalitions. Therefore, approximations and model-gnostic versions are the norms for this method.

Additionally, it can be hard to interpret the results of the Shapley values because they are not fundamentally sparse. Unlike counterfactuals, Shapley values tell us compounded information about the features. The information that Shapley values could be described as the amount of contribution to the prediction for the given instance compared to the dataset's average prediction.

▶ **Definition 2.4** (Shapley value)**.** *Let x be a feature set of length p, where each feature value*

is a player in the game. Then

$$\phi_j = \sum_{D \subseteq \{1,...,p\} \setminus \{j\}} \frac{|D|!(p - |D| - 1)!}{p!}(F(D \cup \{j\}) - F(D))$$

gives us the Shapley value of the feature x_j. Function F gives us the prediction of the model f marginalized over feature values that are missing in the feature set D and is defined as

$$F(D) = \int f(x_1, ..., x_p) dx_{i \notin D} - E(f(x_1, ..., x_p)).$$

This is the original version, which does not scale well with the number of features because it has exponential complexity, so versions using approximations are used. The estimation can be implemented using the Monte-Carlo sampling [11].

Algorithm 9 Monte Carlo sampled Shapley values

Require: m fitted model
Require: T tabular dataset with p rows
Require: $x \in T$
Require: j feature index
Require: N number of iterations
  for i = 1...N **do**
    $z \leftarrow$ random instance from T
    $o(x) \leftarrow$ random function that performs permutation of the features in x
    $x_o \leftarrow o(x)$
    $z_o \leftarrow o(z)$
    $x_{+j} \leftarrow (x_1, ..., x_{j-1}, x_j, z_{j+1}, ..., z_p)$
    $x_{-j} \leftarrow (x_1, ..., x_{j-1}, z_j, z_{j+1}, ..., z_p)$
    $\phi_j^m \leftarrow m(x_{+j}) - m(x_{-j})$
  end for
  $\phi_j = \frac{1}{N} \sum_{m=1}^{M} \phi_j^m$
  return ϕ_j

This version returns results in $O(N)$ where N is the number of iterations and it is possible to improve the approximation by choosing a higher number of iterations.

2.4 Local surrogate (LIME)

As the name suggests, the local surrogate is a local method of explanation, concerned only with one instance from the dataset at a time. It works by creating a model that tries to approximate a model's surrounding behavior f for the given instance x [10].

▶ **Definition 2.5** (Explainable model)**.** *Let f be a fitted model, x instance of the interest from the dataset X, then g is the local surrogate of f obtained by*

$$g = \arg\min_{g' \in G} \mathcal{L}(f, g', \pi_x) + \Omega(g'),$$

where G is the set of all possible local models, π_x defines a neighborhood around the instance, $\mathcal{L}$ is the loss function between models and Ω penalizes the complexity of g'.

The function $\mathcal{L}$ compares the surrogate model g with the original model f. It uses a weighted loss function. The approximation of this method is created by surrounding sample set $\mathbf{Z}$ for the data point x. Then the loss function is weighted by the distance between the $z \in \mathbf{Z}$ and x.

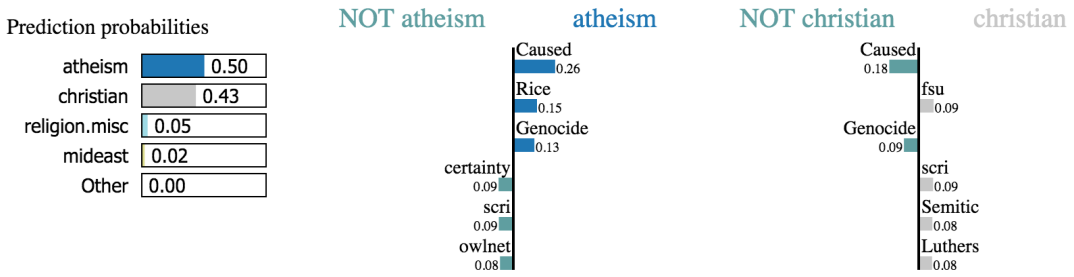


Figure 2.3 Example of a counterfactual from the adult dataset [36].

▶ **Definition 2.6** (Weighted loss function $\mathcal{L}$)**.** *Let f, g be models and $\Pi_x : X \to \mathbb{R}$ be the distance function between the x and z defined as*

$$e^{\frac{-D(x,z)^2}{\sigma^2}}$$

which is the exponential kernel based on cosine distance D. Then the function

$$\mathcal{L}(f, g, \Pi_x) = \sum_{z,z \in \mathbf{Z}} \Pi_x(z)(f(z) - g(z'))^2$$

is the weighted loss function used in the LIME method.

It is apparent that this method is not interested in the implementation details of the model and only works with its inputs and outputs. That means this is a black box method. To fully comprehend the power of this method, two questions need to be answered. What kind of model should the local surrogate be, and how to determine the neighborhood π_x for the tabular data?

The explainability comes from the surrogate model, so it should be interpretable by default. An example of such a model could be the decision tree. Determining the neighborhood π_x for tabular data is more complicated. We cannot simply take training samples around the instance x because there is a high probability (higher than in regression problems) that a given sample will not be valid. One solution introduced in 2.3 creates new instances from the training dataset.

Another solution from the original article works with data samples from the normal distribution around x, but this again needs to tackle the problem of defining the neighborhood of the x. They use an exponential smoothing kernel to define the neighborhood, but how should the size of the kernel be determined?

This model was originally defined for NLP problems. Those problems are referenced in the original paper, but this explanation algorithm is model agnostic. LIME exists for time series, visualizing essential parts of the series data points.

Figure 2.3 shows an example of this model output, where the goal of the model is to predict if a given text is written by an atheist, who is agnostic about the knowledge of god existence.

2.5 SHapley Additive exPlanations (SHAP)

SHAP tries to combine the Shapley values method with the local surrogate model. It computes the Shapley values, then defines the explanation as an additive linear model of feature contributions, which is highly explainable. That detail makes SHAP a local surrogate model.

▶ **Definition 2.7** (Additive model g)**.** *Let $z \in \{0, 1\}^N$ be a coalition vector, M the maximum coalition size and ψ is the feature attribution for a feature j, then SHAP explanation is a method*

defined as

$$g(z') = \phi_0 + \sum_{j=1}^{N} \phi_j z'_j.$$

The coalition vector represents the coalition set from the Shapley values section. The SHAP paper defines two versions of this method. There are valuable properties when using this method that is not available in other additive methods [38].

▶ **Definition 2.8** (Local accuracy)**.** *Let f be the model to explain, g the explanation model and x'_z is the data point x filtered using the coalition vector $z \in 0, 1^N$, if $z = N$, which means that no feature is filtered from x. Then*

$$f(x) = g(x').$$

That means given the whole data point $g(x)$ behaves as $f(x)$.

Local accuracy expects the explanation model g to match the model to explain f in the case where x is equal to x'.

▶ **Definition 2.9** (Missingness)**.** *Let x'_z be the filtered data point of x and g be the explanation model. If $x'_i = 0$, then $\phi_i = 0$, which means missing data does not affect the model.*

Missingness states that if the given feature is missing from the data point x, then it should not have an impact on the model at all.

▶ **Definition 2.10** (Consistency)**.** *Let f be the model to explain, g the explanation model, h transformer that takes z coalition vector and returns x without features where $i \in (0, z), i = 0$ and z i denotes setting $z_i = 0$. Then for any model f, g, if*

$$g(h(z)) - g(h(z\ i)) \geq f(h(z)) - f(h(z\ i))$$

for all inputs z, then

$$\phi_i(g, x) \geq \phi_i(f, x).$$

The third one deals with the fact that if the model changes such that the contribution of the input i increases, then its ϕ_i should not decrease.

The paper proves that only one possible explanation model g satisfies the properties and follows Definition 2.7. The values ϕ_i are Shapley values introduced in Section about them [38]. Figure 2.4 contains example visualizations used with this method. The SHAP method is both global and local and can help explain a single prediction or the overall behavior of the method.

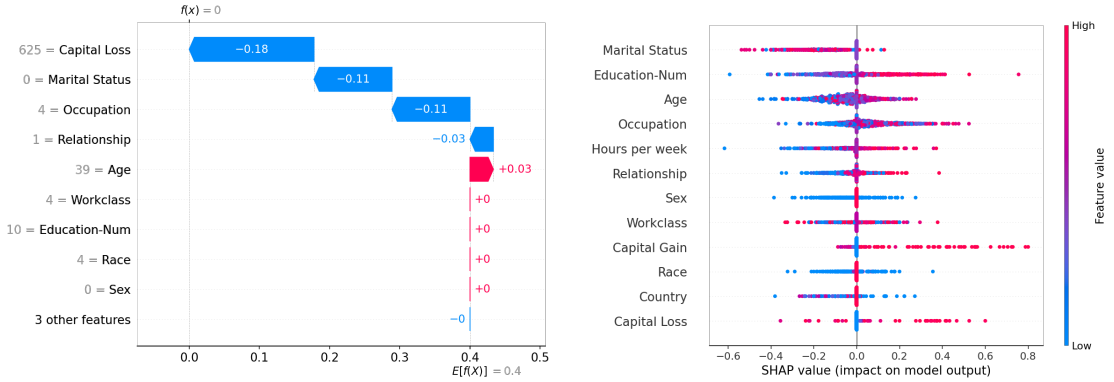
The model then goes on and defines different approximations and versions of the SHAP method. The KernelSHAP version is model agnostic and works with any model with a prediction function. Additionally, TreeSHAP is for tree-like models and Linear SHAP for linear models is presented. TreeSHAP will not be used for explanation in this thesis, even when tree-like classifiers are used, because of problematic interpretability.

2.5.1 KernelSHAP

KernelSHAP is a more general algorithm from the SHAP paper [38]. It uses the SHAP kernel, which is a kernel, that gives us Shapley values as coefficients when applied.

▶ **Definition 2.11** (SHAP kernel)**.** *Let N be the maximum coalition size and z' coalition vector then SHAP kernel is defined as*

$$\pi_x(z') = \frac{(M-1)}{\binom{M}{|z'|}|z'|(M-|z'|)}.$$



(a) Local explanation of which features contributed to the classification of $f(x) = 0$.

(b) Global explanation where the color of the dot corresponds to the feature value and x-axis position defines the impact. Every data point in the data set is represented as a dot.

Figure 2.4 SHAP explanations on the census dataset [36].

Having $\pi_x(z)$ and the model $g(z)$ it is time to train the model. It is done by optimizing the given loss function L:

▶ **Definition 2.12** (SHAP loss function L)**.**

$$L(f, g, \pi_x) = \sum_{z' \in Z} [f(h_x(z')) - g(z')]^2 \pi_x(z').$$

This loss function is called squared error. It is frequently used with linear regression. It is possible to improve this loss function by regularization terms used for the linear regression.

The definition still needs to mention the function $h_x(z)$. The original model f does not work with the coalition vector, as is the case for g, so we need to convert the coalition vector to an instance x. The function $h_x(z)$ does that by keeping original values for features in the coalition and randomly replacing the rest from the training dataset. That could cause problems if the features are not independent. If that is the case, non-representative instances could emerge from the dataset, skew the explanation and confuse the explainer.

2.6 LOFO - Leave One Feature Out

Ahmet Erdem introduced this simple method for computing the feature importance on the dataset [39]. It works by iteratively removing one feature from the dataset in every iteration, then cross-validates to better understand the importance of the feature. The algorithm is model, score metric, and cross-validation agnostic. A different metric, like the categorical cross-entropy for the classification problems, is possible. The default cross-validation used in the method is the K-Fold cross-validation.

▶ **Definition 2.13** (K-Fold cross-validation)**.** *Let* $\mathbf{X}$ *be a dataset with* n *data points and* $k \in \mathbb{N}$*, then* $k - Fold$ *cross-validation shuffles the data-set and creates* k *cross-validation dataset pairs* (d_{train}, d_{test}) *on which the estimator is trained and then evaluated. The result of the K-Fold cross-validation score is equal to the average between the evaluations.*

Algorithm 10 showcases the sequential version of this method. The method is naively parallel because cross-validation steps are independent. It returns both the mean score and its standard deviation, and there is a normalization step where the score from the dataset without the f feature is subtracted from the base score (score on the whole dataset).

Algorithm 10 Compute importance for LOFO

Require: $\mathbf{X}$ dataset of a size n
Require: $s : X \to \mathbb{R}$ scoring function
Require: $e : X \to Y$ estimator
Require: $c : \mathbf{X} \to (\mathbf{X}_{\text{test}}, \mathbf{X}_{\text{train}})$
Ensure: score

base_score $\leftarrow cv(\mathbf{X}, s, e, c)$
features $\leftarrow \mathbf{X}_{\text{features}}$
for f in features **do**
    $\mathbf{X}_{no\_f} \leftarrow$ leave$(\mathbf{X}, f)$
    score$_f \leftarrow cv(\mathbf{X}, s, e, c)$
end for
normalized_scores $\leftarrow base\_score - score_f$ for each f
score$_{\text{mean}} \leftarrow$ mean$(normalized_scores)$
score$_{\text{std}} \leftarrow$ std$(normalized_scores)$
return score

Chapter 3
Proposed methods

The chapter contains descriptions of the methods proposed in this diploma thesis. Those are methods leveraging the ARIMA models used mainly in forecasting. Additionally, an extension of the LIME method using the local surrogate derived from the image-based method is proposed.

After the thorough literature review in the last chapter, it is time to propose new methods. The methods presented in this thesis are primarily based on the autoregressive and moving average methods for forecasting, which means predicting the following values based on historical data. Those models can also be used for fitting existing data by comparing them with the original time series.

Three models are going to be presented. The simplest one is the AR model, where the dataset features are equal to the coefficient values ϕ. The second and third models use a more powerful ARIMA model, with different ways of choosing the model order. The first uses the AutoARIMA technique, while the second uses the grid search algorithm and tries many configurations.

3.1 ARIMA coefficients as time series features

In the paper from Mochaourab et al.[9] they used Fourier and wavelet transforms to generate a tabular dataset. It is possible to do a similar thing using ARIMA model coefficients as the features in the table. Evidence shows that this feature extraction is competitive against other models [40] in the classification problem. The version described in the series combined ARIMA models with interval-based classification methods. Doing so improved the classification performance, but the explainability of the model is now more difficult. The reason is that humans are better at grasping fewer similar features.

3.1.1 Autoregressive Integrated Moving Average model

The ARIMA model, which combines AR and MA models and solves detrending through integration [41], will be presented in this section. The definitions for these simpler models will be laid out, followed by an introduction to the ARIMA model.

▶ **Definition 3.1** (AR(p) model)**.** *Let X_t be a time series, then the model defined as*

$$X_t = c + \phi_1 X_{t-1} + \phi_2 X_{t-2} + \cdots + \phi_p X_{t-p} + \varepsilon_t$$

is an autoregressive model with order p, intercept c, and error ε_t.

Order p defines several past values considered in the prediction of actual value X_t. It is equal to the regression model with lagged time series values as the features. The $AR(p)$ is used as a simpler classifier version of the $ARIMA(p, d, q)$ model in this thesis. There is additionally a random term ε, which is used to represent the uncertainty of the model. It should preferably be equal to white noise.

An example of the AR(p) model can be the AR(1) model, which has a form defined as

$$X_t = c + X_{t-1} + \varepsilon_t.$$

This model with the $\phi_1 = 1$ is called the random walk model because it describes the process of the same name.

▶ **Definition 3.2** (MA(q) model)**.** *Let X_t be a time series, then the model defined as*

$$X_t = c + \theta_1\varepsilon_{t-1} + \theta_2\varepsilon_{t-2} + \cdots + \theta_q X_{t-q} + \varepsilon_t$$

is a moving average model with order q, intercept c, and error ε.

Order in the MA model defines how much should previous p random element influence the current prediction X_t.

▶ **Definition 3.3** (Delta operator)**.** *Let X_t be a time series. Then the delta operator does differencing using the following formula*

$$\Delta X_t = X_t - X_{t-1}$$

This operator can be applied d times depicted as $\Delta^d X_t$.

It is preferable to have stationary time series because if a trend or seasonality model uses previous values, as in the case of $AR(p)$, its prediction becomes harder. Using the Delta operator, the ARIMA model leverages differencing to accomplish a weak stationary time series. If there is an upward or downward linear trend, differencing could help. Modern implementations use algorithms that work even with non-linear trends.

▶ **Definition 3.4** (ARIMA(p, d, q) model)**.** *Let X_t be a time series, and X_t^d its Δ^d differentiation. Then model defined as*

$$\begin{aligned} X_t^d &= c + AR(p) + MA(q) + \varepsilon_t \\ &= c + \phi_1 X_{t-1} + \phi_2 X_{t-2} + \cdots + \phi_p X_{t-p} + \theta_1\varepsilon_{t-1} + \theta_2\varepsilon_{t-2} + \cdots + \theta_q X_{t-q} + \varepsilon_t \end{aligned}$$

is called ARIMA model with order p for its autoregressive part, order q for the moving average part and d number of differentiations.

The strength of the ARIMA model is mainly in the field of forecasting. but coefficients could also be used as features for classification [40, 42].

A time series dataset is converted to a tabular dataset by creating an ARIMA model for every observation and taking its coefficients. The number of features is determined by the maximum value of p and q in the dataset + 2 for the number of differentiations and the intercept. If there are differences between the dataset records in terms of p and q orders, we can fill orders that are not used for the given record with 0.

One last piece needs to be included in making this method relevant and comparable to feature extraction using Fourier and wavelet transforms. How should the ARIMA model's orders p and q be determined? Strategies already exist to determine the order automatically by effectively doing a step-wise grid search. The algorithm is available in the R language `forecast` package [43]. The order estimation is done by combining unit-root tests with Akaike information criterion.

The unit-root test checks that the root of the characteristic function for the $AR(p)$ process is a unit root (equal to 1). When that is the case given process is non-stationary and should be differentiated. The most common unit-root test is called the Augmented Dickey-Fuller test or ADF.

▶ **Definition 3.5** (Augmented Dickey-Fuller test)**.** *The Augmented Dickey-Fuller is a statistical test defined as*

H_0 *There is a unit root in the characteristic function.*

H_a *The time series is trend-stationary.*

Trend stationarity means that after considering the trend in the time series (for example, by differentiating), the time series should be stationary. Akaike criterion is a metric used to evaluate model complexity. Lower order ARIMA processes are preferred because there is a possibility that higher orders would sabotage each other.

▶ **Definition 3.6** (Akaike information criterion)**.** *Let k be the number of parameters for the given model and L its likelihood. Then Akaike information criterion can be computed using*

$$AIC = 2k - 2ln(L),$$

where ln is the natural logarithm.

▶ **Definition 3.7** (Akaike information criterion for ARIMA)**.** *Let $ARIMA(p, d, q)$ be an model and L its likelihood of the model fitted on the integrated data, then we can compute the AIC using the following formula*

$$AIC = 2(p + q) - 2ln(L)$$

.

After preprocessing the time series dataset to a tabular dataset, general classification algorithms such as Support Vector Machines or Gradient Boosting can be utilized. Since the dataset has been converted into tabular data, any explainability method applicable to tabular data can be employed. Counterfactuals and SHAP values will be attempted in this regard.

3.1.2 LIME explanations with the ARIMA models

LIME explanations are method independent, and the only thing that needs to be taken care of is the explainable model g. There is no model available for time series data currently. The model used for locally approximating tabular data was tried for time series. It can be done by converting the time series into tabular data using ARIMA or ROCKET classifier. This strategy did not pan out because it is hard to perturb the features while ensuring the validity of the observation. It is possible to use swapping of features between different observations. However, even after solving this case, it is highly questionable if the results will be more interpretable than SHAP for tabular data. One could also create tabular data from the time series itself, but this comes at the price of a huge performance loss, and the methods have problems when features are correlated. That is almost always true in the case of time series data.

The second possibility was to use the model for locally approximating image data. An image is effectively multivariate time series data, and time series can be represented using a 1-D image. The method that is used with pictures already perturbates surrounding pixels, which works better with highly-correlated datasets.

▶ **Definition 3.8** (Transformation of the time series to an image)**.** *Let X_t be a time series, then $f : X_t \to I^{1,n}$ is a transformation of a time series to an image representation containing only the gray scale channel, where the ith value $I^{1,i}$ is equal to $x_i \in X_t$.*

The method generates images close to the original with some pixels turned off by setting them to 0. It then computes the distance between the images and fits a classifier for each image instance. Those classifiers are then weighted based on the Euclidean distance between

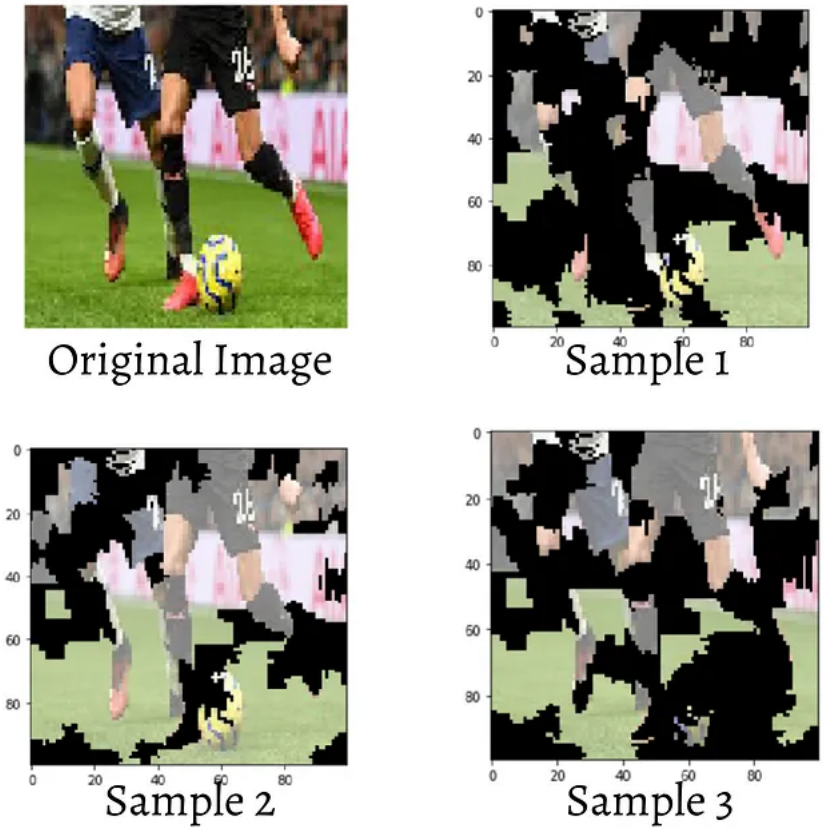


Figure 3.1 Original image, and LIME generated neighboring instances

the original and the new image. A simple method like linear regression is used to classify, and feature importance is coded in the weights.

In Figure 3.1, it is possible to see the original image with some of its neighboring instances. These images can be grouped into the new training dataset for the explainability model. Figure 3.2 shows the output from this explanation method. It tells the observer which parts of the image are essential for correctly classifying it.

3.2 ROCKET method for time series classification

The ROCKET method is a powerful state-of-the-art predictor that uses convolutional kernels for feature selection. The problem is that the kernels are generated randomly with little to no insight into what features they select. It would be beneficial to have a performant predictor like this, which is also explainable. Therefore, the explainer introduced in the last chapter will be used with this model.

The original version of this method uses ridge regression, which cannot return probability distribution for the classes. For that reason, simple logistic regression was used instead of ridge regression. Both models have similar results that could be additionally tuned, for example, using a lasso penalty in the case of logistic regression.

The model remains the same as described in the original paper [8]. It first transforms the data using the rocket transformer and then scales the features using z-normalization. Transformed

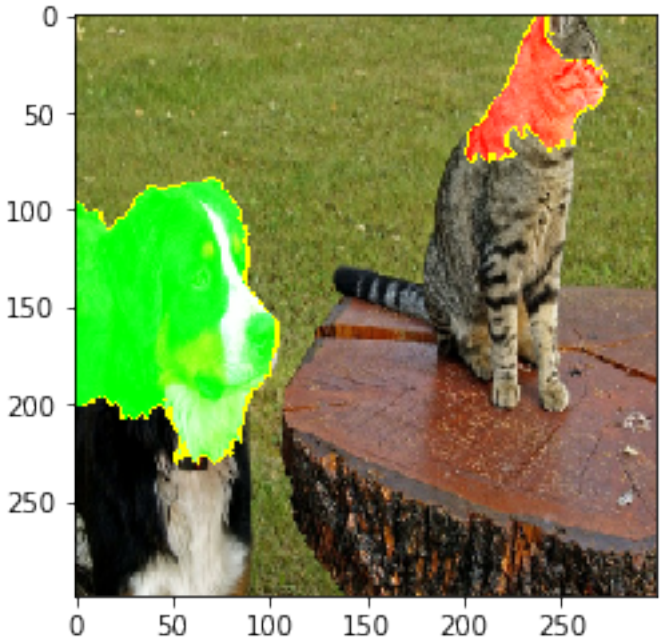


Figure 3.2 Explanation from the LIME method. Colors define which parts of the image were used to classify and which category (dog, cat).

data is then fed to the logistic regression to generate the class probabilities. Unlike ARIMA models, there are no easily accessible methods for interpretation for the ROCKET method. The main reason for this model implementation is to figure out how to explain it. The explanation will be supplied using the LIME method described in the chapter above.

Chapter 4

Implementation

This chapter will describe the implementation details of the methods introduced above . The methods were implemented using `Python 3.11`, and the libraries used for each method will be mentioned when their usage is described. The implementations were tested, and their performances were measured in a white-box phase on the ArrowHead and Coffee datasets.

Methods implemented in this thesis are trying to be highly reusable. For that reason, standardized interfaces known to the Python community are used. All implementations are one of two kinds. Classifiers are full-fledged models that can be used for the whole train/test pipeline. Unlike transformers, they can be used out of the box for classifying time series. Those transform the raw time series dataset to some other feature space. Both of these kinds have rigid protocols in a battle-tested library called scikit-learn [13]. Datasets can be passed to the classifiers and transformers using numpy and pandas data formats.

Classifiers use the fit method to learn from the training dataset and then predict the class using the predict_proba method. The result of the predict_proba is equal to the probability of each class being the correct answer. By defining these two methods, scikit-learn provides additional ones when using its ClassifierMixin. The classifier mixin contains methods like predict, which returns the most probable class, or score that computes the accuracy score given the correct class labels.

Transformers also use the fit method to prepare the dataset for transformation. After that, the dataset can be transformed using the transform method. Again by leveraging the TransformerMixin, additional features can be obtained. For example, a helper function fit_transform does both steps simultaneously on the same data.

Multiple transformers can be chained together using the Pipeline class in the scikit-learn library. This way, the user can ensure that train and test data are not mingled. Using pipelines creates a much smoother API for training and testing models. There should be precisely one classifier or regressor at the end of the pipeline.

Scikit-learn contains implementations for tabular classification models and supporting functions for general time series classification. The random forest classifier implemented in this library was used for the ARIMA models. Listing 1 contains definitions in the Python implicit protocol format.

Apart from scikit-learn, the thesis uses these additional libraries to implement the explainers. Most of them are scikit compatible and implemented as transformers or classifiers. Python's whole data science ecosystem uses numpy and pandas libraries to represent tabular and array-like data. Other libraries leverage operations from these libraries to implement the models. Those libraries are listed below. Their usage will be discussed in detail whenever they underline the models described in this thesis.

Statsmodels Contains statistical models for time series analysis. These are used for the ARIMA classifier. The implementation is slow even for small time series, which means that it is not suitable for transforming a time series dataset containing a huge amount of observations [44].

Pmdarima Is a port of the library pmdarima from R language, which contains the auto_arima function. This function helps estimate the order for ARIMA models using the maximum likelihood [43].

Statsforecast Better version of the Statsmodels, which is much faster and centralized solely around time series forecasting and analysis. It contains a custom implementation of AutoARIMA, which is faster than pmdarima. When testing, Statsforecast managed to be even 7x faster per time series, than Statsmodels combined with Pmdarima [45].

PyTS Python time series classification library containing most of the algorithms described in the Literature review chapter [46].

CatBoost Modern gradient boosting library, which is scikit-compatible [47].

TsFresh Contains many transformers that can extract features from the time series. These were used for the Catch22 algorithm [48].

LIME Implementation of the local surrogate explainability model. It contains other explainers for problems like tabular, image, or text classification[10].

SHAP Implementation of the SHAP explainability model. It contains the logic for computing Shapley values and visualizations, which can be seen below [38].

Sktime Comprehensive library for everything time series related. Supports a wide variety of tasks from forecasting to clustering and classification. It also contains scikit-learn compatible interfaces and pipelining of different models and transformers [49].

Arff Marshals and unmarshals time series data in the arff format used by the UCR archive. It can contain additional metadata and description for the time series dataset. The result of the arff marshaling is a pandas dataset.

All the libraries are installed using the Poetry package manager, and details on how to run the notebooks is in the git repository included with the thesis.

4.1 FFTTransformer

The FFTTransformer is the fast Fourier transform of the dataset implemented according to Mochaourab et al. [9]. It takes a time series dataset pandas or numpy formats and produces two features per every FFT coefficient. Features are the amplitude and the angle of the complex number.

▶ **Definition 4.1** (Computing FFT amplitudes)**.** *Let X_t be a time series of a length t, then amplitude A_t can be computed using the following formula*

$$A_t = \frac{2}{t} \cdot abs(rfft(X_t)),$$

where rfft is the real fast Fourier transform and abs is the absolute value function.

The transformer does not use additional high-level libraries like sktime or statsforecast as seen in Listing 2. There is the computation of both features from a real fast Fourier transform. It is fully implemented using the numpy library, which makes the transformation time effective. The transformer pipelined with the RandomForestClassifier to create a complete model for classification.

```
from typing import Protocol, Self
import pandas as pd
import numpy as np

class Transformer(Protocol):
  def fit(self, X: pd.DataFrame) -> Self:
    pass
  def transform(self, X: pd.DataFrame) -> pd.DataFrame:
    pass

class Classifier(Protocol):
  def fit(self, X: pd.DataFrame) -> Self:
    pass
  def predict(self, X: pd.DataFrame) -> np.ndarray:
    pass
  def score(self, X: pd.DataFrame, y: np.ndarray) -> np.float64:
    pass
```

Code listing 1 Transformer and classifier protocols

```
class FFTTransformer(BaseEstimator, TransformerMixin):
  def __init__(self, n=11):
    self.n = n

  def fit(self, X):
    pass

  def transform(self, X):
    samples_n = X.shape[1]
    f = rfft(X, n=self.n)
    ampls = 2 / samples_n * np.abs(f)
    angle = np.angle(f)
    concat = np.concatenate((ampls, angle), axis=1)
    return concat

  def fit_transform(self, X, y=None, **fit_params):
    return self.transform(X)
```

Code listing 2 The FFTTransformer

Table 4.1 Time series data-frame from the coffee dataset

0	1	2	3	...	283	284	285
-0.518419	-0.485884	-0.505007	-0.560183	...	-1.933631	-1.934964	-1.936007
-0.548462	-0.533681	-0.514723	-0.559256	...	-1.940727	-1.943786	-1.944308
-0.472634	-0.415546	-0.359929	-0.430496	...	-1.989143	-1.990659	-1.992301
-0.509521	-0.484218	-0.477951	-0.511287	...	-1.950172	-1.951748	-1.952922
-0.563427	-0.533896	-0.543822	-0.598246	...	-1.866945	-1.868419	-1.869571

4.2 ARTransformer

Every ARIMA model can be converted to $AR(p)$ model using polynomial division [50]. So this method leverages that and computes AR coefficients only. Doing that gives a significant performance boost for both the training and test phases.

TSFresh library is used to compute coefficient values. It contains ar_coefficient function, which takes p value and returns first p coefficients of the model. The important question is, how should the order p be picked? One of the ways is by using the hyper-parameter tuning algorithm like GridSearch. This strategy was used in the implementation. Because only one parameter is optimized, the grid search performance is good. GridSearch can be seamlessly combined with the classifiers and transformers using the pipeline feature from the scikit-learn.

This method achieved an accuracy of 0.78 on the Coffee dataset but had problems correctly classifying arrows, with an accuracy score of only 0.46 for that dataset. The reason could be that the arrow dataset is better explained using shapelets because the classes are mostly undistinguishable, except for small patterns. The AR model is better suited to represent the global behavior of the series, which is the case for the Coffee dataset.

4.3 ARIMAClassifier

The ARIMAClassifier is the `scikit-learn` compatible class with an implementation based on the chapter 3.1.1. It is effectively a wrapper around the `RandomForestClassifier` from the `scikit-learn`, which transforms time series data from the time domain to ARIMA model features using the `arima_tabular` function shown in 3. The function uses `auto_arima` function from the `pmdarima` library. It ignores the seasonal component because `auto_arima` guesses orders for a more robust model called SARIMAX. After those model parameters are taken from each dataset time series and written to a pandas dataset, which is then returned.

Tables 4.1 and 4.2 show the datasets before and after processing. The SHAP library then uses our newly created classifier and the data in the later format to do the explanation. There are multiple ways used to visualize SHAP values. More on that in the Results chapter. It seems like values after preprocessing are sparse for this given dataset. It could be used with faster and more memory-efficient matrix formats.

After trying this method of transforming the time series, it was apparent that the sparsity and different orders of the time series data are causing problems. Doing white-box analysis on the coffee dataset showed us that the differences between orders of various series can be huge. The accuracy of the Coffee dataset was 55%, and on the Arrow dataset, it was 32%. Additionally, computing orders for every record in the dataset costs significant time. It took the Coffee dataset six minutes to do one train to test iteration.

Taking n random series from the dataset makes it possible to estimate the average order as the mean of the n orders. Then there is no need to generate the orders for every series. All the series are going to use the averaged order, which gives the model significant performance for both speed and accuracy. Only half the time was needed to fit and test both datasets, and their

Table 4.2 Coffee dataset after preprocessing

intercept	integration	ar0	ar1	ar2	ma0	ma1	ma2
0.0	1.0	0.733665	0.000000	0.000000	0.177208	-0.128667	0.285673
0.0	1.0	0.190291	0.078586	0.266296	0.745033	0.000000	0.000000
0.0	1.0	0.151328	0.453973	0.000000	0.902177	0.000000	0.000000
0.0	1.0	0.840085	0.000000	0.000000	0.178405	-0.406996	0.000000
0.0	1.0	0.551166	0.000000	0.000000	0.454275	0.019579	0.369809

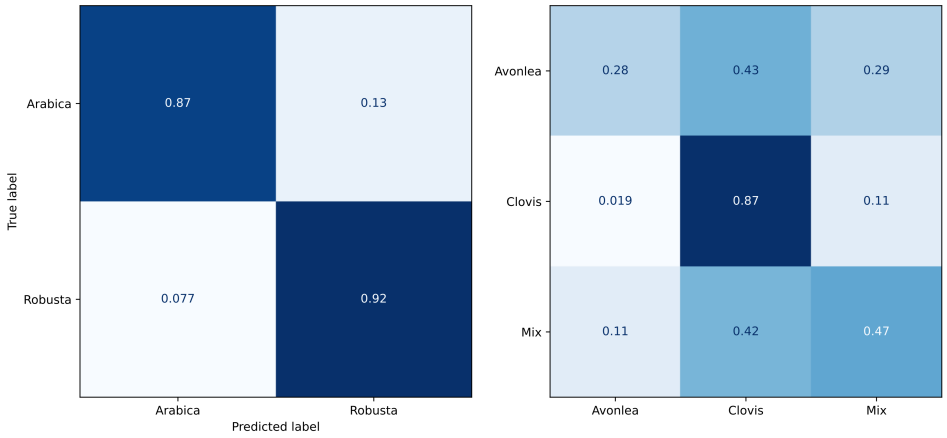


Figure 4.1 Confusion matrix of the ARIMA classification model

accuracies rose to 41% resp. 64%.

An alternative to the `auto_arima` could be adding search on hyperparameters to the pipeline using the `Grid` or `Randomized` search from the `scikit-learn` library [13]. Fine-tuning using the `RandomizedSearchCV` raises the accuracy of the model to 89% for the Coffee dataset and to 51% for the ArrowHead dataset. Only the ARIMA orders in the range $(0...3, 0...1, 0...3)$ were considered, which means that the search is swift.

The `RandomizedSearchCV` is used in place of the `GridSearchCV` because it is faster. The GridSearch tries every possible setting of the hyper-parameters, unlike `RandomizedSearchCV`, which samples different hyper-parameters from the distribution defined at the start. It then uses 3-fold cross-validation to pick the best one from the samples.

From the resulting confusion, a matrix can be found in Figure 4.1, and it is possible to see that prediction on the Coffee dataset works well. Classifier struggles on the ArrowHead dataset and shows positive bias towards the `Clovis` shape. There are further steps in the preprocessing phase to increase the classification power of the model. For example, the `ArrowHead` dataset has a shapelet in the time series's $(60, 190)$ interval that could be used instead of the whole series. After this change to the time series, the accuracy improved to 57% on the dataset.

Apart from the `RandomForestClassifier` as the tabular classifier `CatBoost` was also tried giving overall better performance for longer learning times. It is a gradient boosting algorithm,

```
def arima_tabular(X):
  models = []
  for x in X:
    models.append(auto_arima(x, stepwise=True, seasonal=False))

  max_ar = max([len(m.arparams()) for m in models]) max_ma =
  max([len(m.maparams()) for m in models])

  features = [] for m in models: if m.with_intercept: intercept = m.params()[0]
  else: intercept = 0 feature = [intercept, m.order[1]]
  feature.extend(np.pad(m.arparams(), (0, max_ar - len(m.arparams()))))
  feature.extend(np.pad(m.maparams(), (0, max_ma - len(m.maparams()))))
  features.append(feature) return np.array(features)
```

Code listing 3 The `arima_tabular` function

a faster version of the more known `XGBoost` method.

Lastly, another library for auto arima functionality was tried called statsforecast. This library is highly optimized, and its auto ARIMA feature can be even 10x times faster than the one using the pmdarima. Still, the grid search version of the algorithm works the best, albeit it is slower than using this method. The version using the statsforecast implementation was developed as a transformer, so combining it with a range of different classifiers is possible.

4.4 LimeTimeSeriesExplainer

The LimeTimeSeriesExplainer is a method-agnostic explainer compatible with the scikit classifier APIs. It takes care of both transformations of the data point from the time series dataset and training on the explainable model needed for the explanation. Under the hood, it uses the python lime library [51].

The explainer takes the model and the explanation data point and returns a result. This can be further used to plot essential parts of the time series, which helped the original classifier arrive at the current classification. The solution is available in Code Listing 4. Images generated by the lime contain three channels for RGB, so the first step is to convert the given image only to 1 channel grayscale, which is equal to the original time series. An empirical conversion function using the dot product and coefficients is used, but the sole average could also be leveraged. The plot function on the result class plots the original time series and the crucial parts as colored backgrounds. The output can be seen in Chapter 5 (Experiments). The last mask is colored gray and represents unimportant parts of the time series. This version of the plotting algorithm expects only two classes, but the final version generally works on any number of classes.

The plot function leverages the image explainer function, which returns the mask for the image, where each image pixel is set to the classification value from which the original model derived its classification. This explainability model can be used both with the FFTTransformer algorithm and the ARIMA models, so their explanation differences will be compared in the next chapter to see if they give similar and relevant results.

4.5 RocketExplainer

The ROCKET explainer is implemented as a pipeline, which firstly transforms the dataset into a shape supported by the rocket transformer in the sktime library [49]. The transformers in

```
class LimeTimeSeriesResult:
  def __init__(self, x, explanation):
    self._x = x
    self._explanation = explanation

  def plot(self, ax=None):
    t_n = self._x.shape[0]
    temp, mask = self._explanation.get_image_and_mask(
        0, hide_rest=False, positive_only=False
    )
    x = np.arange(0, t_n, 1)
    if ax is None:
        ax = plt.gca()
    y_min = np.min(self._x)
    y_max = np.max(self._x)
    ax.plot(self._x)
    for i, lab in [(1, "positive"), (-1, "negative"), (0, "neutral")]:
        kwargs = {
            "label": lab,
        }
        if i == 0:
          kwargs["color"] = "gray"
        ax.fill_between(x,y_min, y_max,
            where=mask.reshape(1, -1)[0] == i,
            alpha=0.15 if i == 0 else 0.5,
            **kwargs
        )
    ax.legend()


class LimeTimeSeriesExplainer:
  def __init__(self, model, **kwargs):
    self.model = model
    self.image_explainer = LimeImageExplainer(**kwargs)

  def explain_instance(self, x) -> ImageExplanation:
    def _predict_proba(x):
        gray_img = np.mean(x[0], axis=2)
        gray_img = np.expand_dims(gray_img, axis=-1)
        return self.model.predict_proba(gray_img.reshape(1, -1))

    with warnings.catch_warnings():
        warnings.simplefilter("ignore")
        e = self.image_explainer.explain_instance(
            x.reshape(-1, 1), _predict_proba, hide_color=None, batch_size=1
        )
    return LimeTimeSeriesResult(x, e)
```

Code listing 4 The LimeTimeSeriesExplainer and Result implementations

```
X, y, X_test, y_test = get_ds("Meat")

cl = make_pipeline(
    FunctionTransformer(lambda x:  x.reshape(x.shape[0], 1, x.shape[1])),
    Rocket(),
    StandardScaler(),
    LogisticRegressionCV(),
)

cl.fit(X, y)
e = limexp.LimeTimeSeriesExplainer(cl).explain_instance(X_test[0])
```

Code listing 5 Using the rocket pipeline with the explainer. The classifier is trained on the Meat dataset and the Function transformer is used to convert the data into the shape expected by sktime.

the sktime library expect multi-variate time series. There is a need to include an additional dimension to meet the interface, as seen in Listing 5. Then the dataset supported by the sktime library is funneled through the rest of the pipeline.

This pipeline can then be used with the LimeTimeSeriesExplainer in place of the ARIMA coefficient model. The explainer was further modified to allow batch classifications, which makes this method even faster.

The rocket explainer is powerful and able to predict Coffee testing dataset perfectly and ArrowHead with an accuracy of 0.81, without cross-validations. It is expected for the ROCKET classifier to be more accurate than the models based on the ARIMA because ROCKETs convolutional kernels can model both global features and local features like shapelets alike.

Chapter 5

Experiments

This chapter contains experiments both in the field of explainability and classification. It is important to ensure that the methods described in the previous chapter are competitive with the state-of-the-art methods, or at least there is a reasonable trade-off between them.

This section will use the public dataset database available at `timeseriesclassification.com` for a comparative analysis of different models. There are both univariate and multivariate datasets are available in the database, but models will be tested only on a small subset of univariate time series.

The UCR database is optimized for accuracy testing experiments. The time series represented in it are already normalized and otherwise prepared to run classification models on them. For example, ECG signals are split to contain only one heartbeat per time series and labeled as healthy or with myocardial infarction. This way, it is possible to attest to the quality of the time series classification model with the elimination of outside influence.

The right way to test the generality and accuracy performance of the model is to split the dataset into a training and testing split. This way, the possibility of overfitting the data is less likely. Time series in the UCR database are already split to train and test parts. This ensures better comparison between different papers and their methods' performances. In 2017, the group behind the UCR database published a paper colloquially known as the bake-off [15]. Some of the results are taken from that paper because by abiding the process, they should be comparable.

The methods will only be tested on some of the data. Some of the classification problems are enormous, and the classification becomes time-consuming. They were handpicked beforehand without running the methods on them. After that, they were not reconsidered. The first 85 univariate datasets have their accuracy scores available. Those were used in the bake-off paper. The final chosen time series can be seen below.

Coffe Dataset containing spectrographs for the problem of classifying Arabica and Robusta coffee beans. [52]

ArrowHead The arrowhead dataset contains outlines of the images of arrowheads. These outlines are converted using an angle-based method to a 1-D array [6].

Beef Food spectrographs of different beef, from pure beef to beef adulterated with varying degrees of offal. [53]

BeetleFly Contains outlines of beetles and flies for the classification problem of distinguishing between them. [15].

CinCECGTorso Differentiation of 4 people based on their ECG signals using a detector on their torso.

Table 5.1 Datasets selected for the experiment [15]. The first half of the table contains training data statistics and the second half contains the testing data.

Dataset	Samples	Length	Classes
ArrowHead	36	251	3
Beef	30	470	5
BeetleFly	20	512	2
CinCECGTorso	40	1639	4
Coffee	28	286	2
ECG200	100	96	2
EOGHorizontalSignal	362	1250	12
Meat	60	448	3
ArrowHead	175	251	3
Beef	30	470	5
BeetleFly	20	512	2
CinCECGTorso	1380	1639	4
Coffee	28	286	2
ECG200	100	96	2
EOGHorizontalSignal	362	1250	12
Meat	60	448	3

ECG200 Each series contains an ECG signal, and the task is to classify Myocardial Infarction from the normal heartbeat [54]. Heartbeats belonged to only one patient and were extracted from one long ECG series.

EOGHorizontalSignal Contains the horizontal eye movement measured as EOG (electrooculography) signal. There are 12 classes, each representing eye-writing of different Japanese Katakana strokes in the horizontal axis.

Meat Food spectrographs for turkey, chicken, and pork. The task is to classify different types of meat.

General statistics about the datasets can be seen in Table 5.1. None of the time series are of different lengths per sample. Chosen datasets vary in size, number of classes, and samples, respectively. There are also different time series datasets like ECG signals, food spectrographs, or image outlines.

There are some outliers picked for the classification. Those were the CinCECGTorso and EOGHorizontalSignal datasets. The first one contains many observations in the testing dataset, with only a few in the training. This tests the ability of the model to learn from a few shots. Luckily the length of the observation gives plenty of data to learn from. The data represent four different people (classes), and the goal is to differentiate between them. The second dataset contains EOG measurements, which is the corneoretinal standing potential between the front and the back of the eye. The EOG recording device can be used to record eye movement. This is also the purpose of this classification problem. The goal is to detect one of 12 classes representing eye-writing of different Japanese Katakana strokes.

5.1 Accuracy and performance measurements

In the final accuracy examination, 4 models were chosen. These are implemented in the included repository, and three are based on the proposed methods. The FFT model is based on Mochaourab et al. paper[9].

Table 5.2 Comparison between the simple ARIMA and other models. Numbers in the table represent accuracy scores on test dataset splits from the time series classification website [15].

TESTACC	n	FFT	AR	ARIMA	ARIMA-GS	ROCKET	STC	BOSS	TSF	Catch22
CinCECGTorso	4	0.431	**0.752**	0.607	0.652	0.864	0.978	0.915	0.958	0.803
ECG200	2	0.790	**0.800**	0.680	0.760	0.899	0.839	0.878	0.860	0.789
EOGHorizontalSignal	12	0.113	0.185	**0.218**	0.180	0.814	0.760	0.707	0.706	0.673
Meat	3	**0.783**	0.333	0.683	0.533	0.989	0.968	0.981	0.984	0.943
Coffee	2	0.714	0.785	0.892	**0.929**	1.000	0.989	0.986	0.987	0.980
ArrowHead	3	0.360	0.462	0.411	**0.543**	0.859	0.807	0.869	0.797	0.750
Beef	5	**0.466**	0.433	0.200	0.433	0.760	0.736	0.612	0.689	0.473
BeetleFly	2	**0.700**	0.450	0.500	**0.700**	0.885	0.933	0.943	0.833	0.840

FFT Amplitude and Angle from the FFT [9]

AR(p) AR model of the order p

ARIMA(pdq) ARIMA model, orders estimated using AutoARIMA

ARIMA-GS ARIMA model, orders estimated using the grid search

There are many non-stationary time series in the selection, so it is expected that the ARIMA-based models to work poorly on them. This presumption is rendered true as seen in Table 5.2. It contains accuracy scores for proposed methods on classes defined above. The n column contains the number of classification groups. The table is divided into two sub-tables containing models implemented in this thesis and other state-of-the-art models referenced in the research section. Bold emphasis in the first sub-table is used to differentiate the most performant model visually.

The second sub-table contains the ROCKET model, described in the Proposed method chapter, STC, which is the shapelet algorithm introduced in the Literature review, BOSS, Time Series Forests (TSF), and the Catch22 feature extraction method.

The choice of order is an essential factor in determining the accuracy of the ARIMA-based models. It is the main contributor to the difference between those models, as well. Using the model with AutoARIMA order estimation yields worse results than the grid search version but is faster than trying every possible combination of parameters.

Performance comparison results are available in Figure 5.3. It was done on a Lenovo Thinkpad t480s notebook with Intel Core i5-8250U CPU and 24GB of RAM. GPU was not used for training and testing, so it is irrelevant to mention it. Measurement was done on one observation from the testing dataset at a time. Batch optimization could play a significant role, but not every algorithm presented has batch prediction implemented. Measurement was done 30 times and then averaged to rule out additional factors that could affect the time, like scheduling or the CPU load.

Python libraries for time series modeling using ARIMA are generally not optimized for vast amounts of small time series, so there is room for improving the performance. The limits are more implementation based than some underlying problem with the classification design. Even when the GridSearch process does not impact the running time, a distinction can be observed between the ARIMA and ARIMA-GC times. The reason for that is that different core ARIMA models are used. The ARIMA uses newer and more performant statsforecast, unlike ARIMA-GC, which uses the statsmodels version of the method.

Both ARIMA versions have huge performance burdens and speed issues compared to FFT, AR, or ROCKET version. ROCKET is the most competitive one implemented when its accuracy is considered.

Table 5.3 Prediction time for proposed methods in seconds.

Dataset Model	AR	ARIMA	ARIMA-GC	FFT	ROCKET
ArrowHead	0.16	4.76	10.97	0.52	5.40
Beef	0.19	8.54	13.97	0.39	6.51
BeetleFly	0.18	8.04	9.86	0.39	9.69
CinCECGTorso	0.69	9.26	31.88	0.37	12.07
Coffee	0.16	4.28	19.98	0.48	8.62
ECG200	0.20	7.47	2.97	0.40	7.90
EOGHorizontalSignal	0.88	9.74	8.26	0.35	11.11
Meat	0.17	4.72	24.22	0.38	6.45

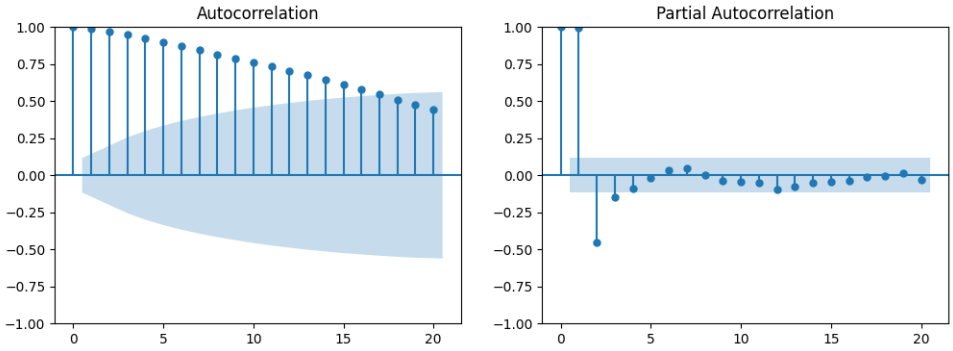


Figure 5.1 PACF of the Coffee dataset sample

5.2 Explainability and interpretability of ARIMA models

This section is concerned with applying explainability models to proposed methods to test their suitability for the problem. SHAPleys additive values and counterfactuals will be used mainly, but ARIMA models are also well-explainable using the PACF and ACF graphs. The LIME method will also tell which parts of the time series are essential for the classification. Lastly, the same method will be used for the ROCKET method, which is much less interpretable by default.

Explaining all the datasets used for the accuracy measurements would be redundant, so the chapter focuses primarily on two series. The coffee dataset was picked because it represents a food spectrogram, which is a common classification problem. The second picked problem is the ECG200 because this type of problem is more concerned with finding shapelets.

The LIME classifier will also be used with the ROCKET dataset to understand the ArrowHead dataset's crucial parts. The critical part will be the outline for the notch part, which distinguishes different arrowhead parts.

5.2.1 AR classifier

The $AR(p)$ is the simplest model proposed in this thesis, so it is reasonable to start with it and build from there. Additionally, $AR(p)$ is an interpretable model because it is possible to derive much about the model behavior from the PACF values.

The graphs in this section represent misclassified samples from the Coffee dataset with an index 0 from the original dataset. By examining the partial autocorrelation in Figure 5.1. The interpretation of the plot is that the first three lags correlate on the 95% confidence interval. That means that to model the time series well, adding higher weights to those lag values is vital.

It could be interesting to compare this simple explanation with the SHAPley additive values

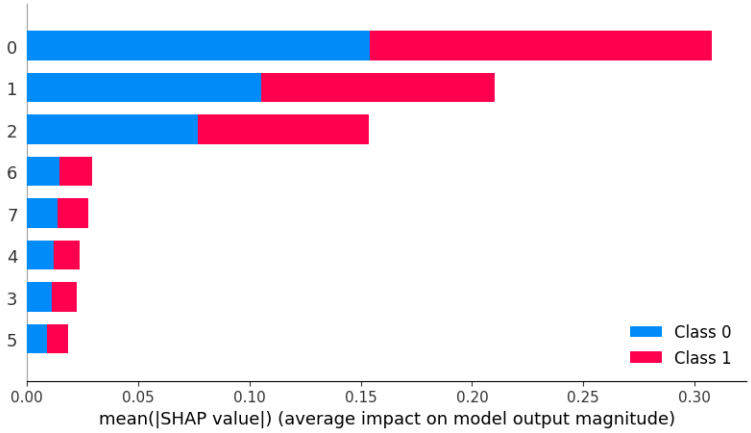


Figure 5.2 AR coefficient sorted by the lag important derived from the SHAP. This is the average importance over the whole test dataset. The y-axis corresponds to the coefficient index.

and the counterfactuals.

5.2.1.1 SHAPley additive values

The SHAPleys additive values are implemented in the Python SHAP library. Multiple types of explainers are described in the 2.5 [38]. The `KernelSHAP` was used for our particular use case because of the interpretability problems of the `TreeSHAP` referenced in the original paper.

Figure 5.2 shows that the most critical lags from the $AR(p)$ model are the first 3, which is on par with our explanation using the PACF coefficients. The reason is that both the explanation and the $AR(p)$ models are additive and linear. More formally, there is a correlation between the coefficients from SHAP with high impact and the lags with high autocorrelation. Using this information to estimate the order of the $AR(p)$ model should be possible. Using the SHAP method, one could create a $AR(10)$ model and filter out unimportant coefficients.

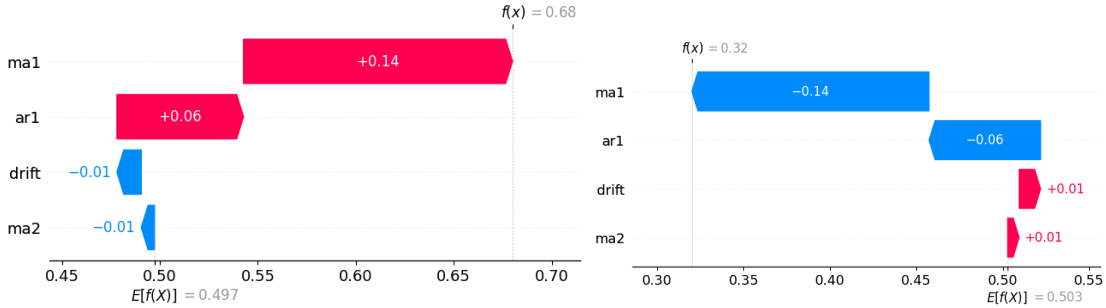
5.2.2 ARIMA classifier

In the case of the AR model, it is easy to deduce the important factors and model order from the PACF. This task is much harder for the ARIMA classifier because the order of the ARIMA models is not evident from PACF and ACF graphs. This section will leverage two methods for the explainability of this problem. They could provide an alternative in choosing the proper order for the ARIMA model.

5.2.2.1 SHAPleys additive values

The explainer was run on the intermediate transformation after extracting the ARMA coefficients because running on the original time series was slow and most of the time, unable to converge using the likelihood estimation of the coefficients. It is still possible to reconstruct the time series from the coefficients simply by running the ARIMA from the start with the new coefficients.

Figure 5.3 shows that the `ma1` affects classification the most. It skews the classification from the expected value of 0.5 to 0.69 and 0.31 respectively. These graphs are concerned only with 1 classification, so this is a local explanation. SHAP allows for global explanations also, which is shown in Figure 5.4. Even though for the concrete explanation, the `ma1` feature was the most important, if we look at the global explanation `ar1` impact is higher.



(a) Explaination for the class 0

(b) Explaination for the class 1

Figure 5.3 SHAP explanations

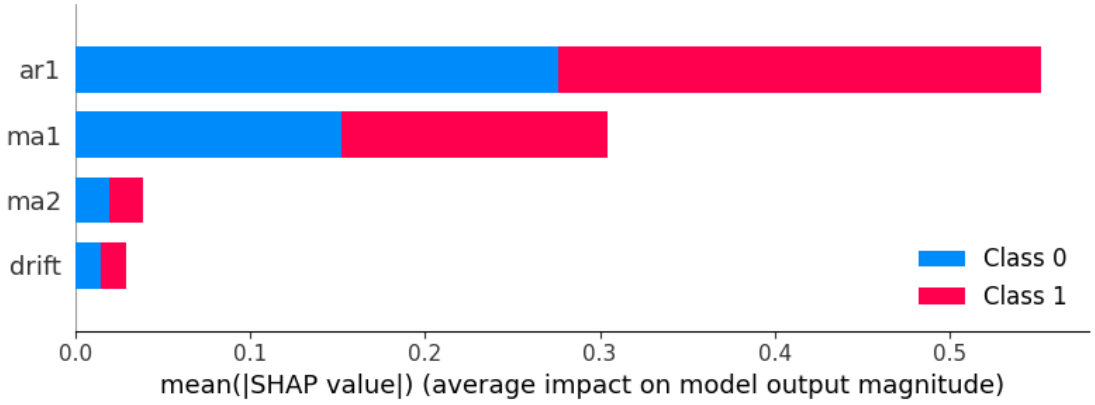


Figure 5.4 Global explanation of the feature importance

The second global explanation in the form of a beeswarm graph in Figure 5.5 confirms that. Higher values for the ar1 correlate, with a high negative impact on the model output.

It is important to remember that *ar* and *ma* coefficients are concerned with lagged values. So the SHAP explanations tell us which lagged values are essential to the model. In our example, the high impact of the $ar1$ coefficient means that there is probably a strong correlation between X_t and X_{t-1} in the context of modeling the time series data.

5.2.2.2 Counterfactual explanations

The counterfactual explanations of the ARIMA model are based on the growing spheres algorithm. It works by perturbating a given feature set and minimizing the number of changes so the results are better explainable.

Problems with this method are found because of the minimization of the perturbations. It creates minimal differences between the original and enemy series and potentially games the predictor as seen in Figure 5.6. The difference between the enemy and the original is only in the `ar1` coefficient in the sparse version. The difference is 0.01, which could be solved by finding the enemy before the time series transformation, but that method is substantially slower than the one after the transformation.

Because of this problem, counterfactual explanations are not as interpretable given the Coffee dataset.

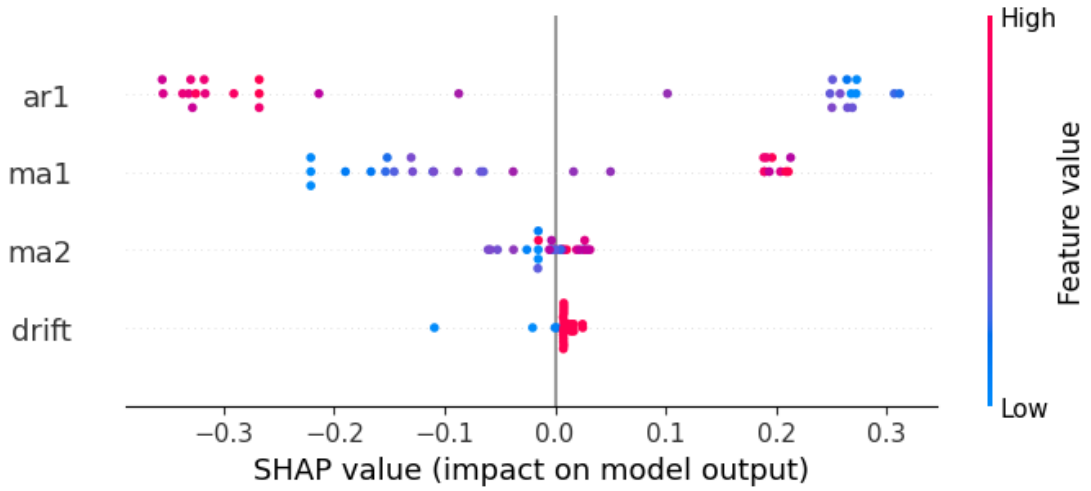


Figure 5.5 Global explanation of the feature importance

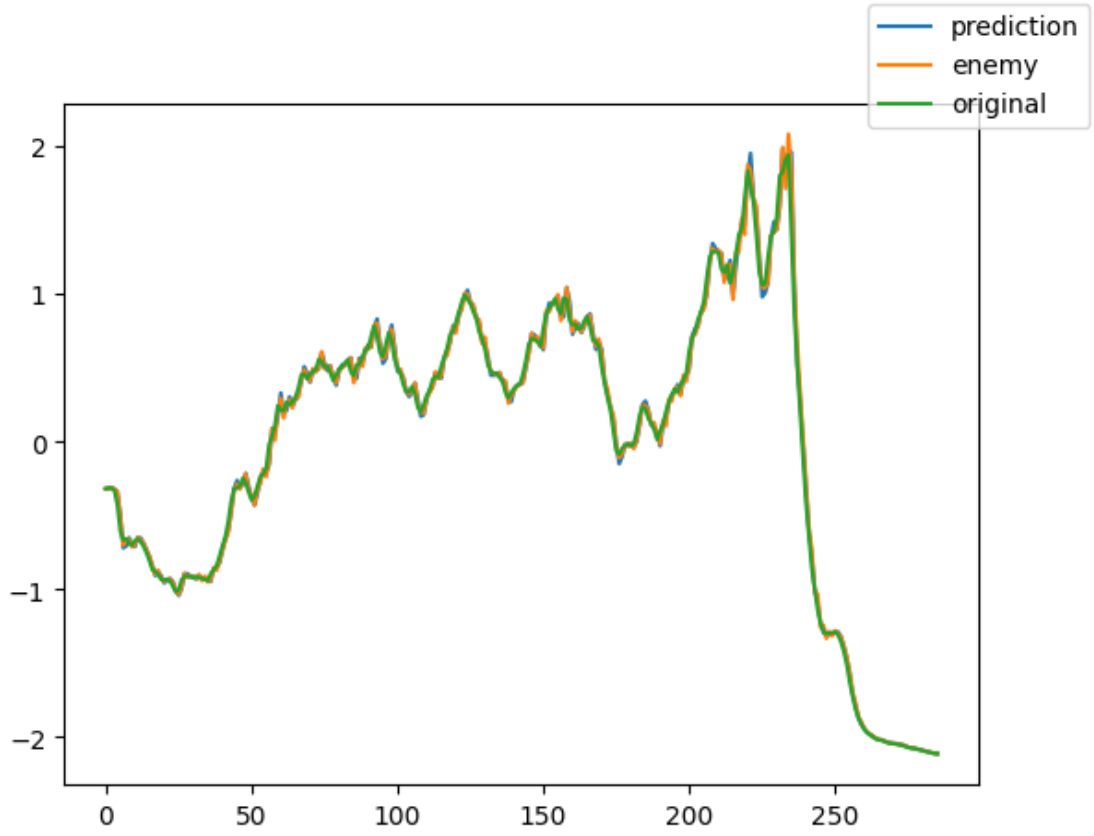


Figure 5.6 Counterfactual explanation for the Coffee datum

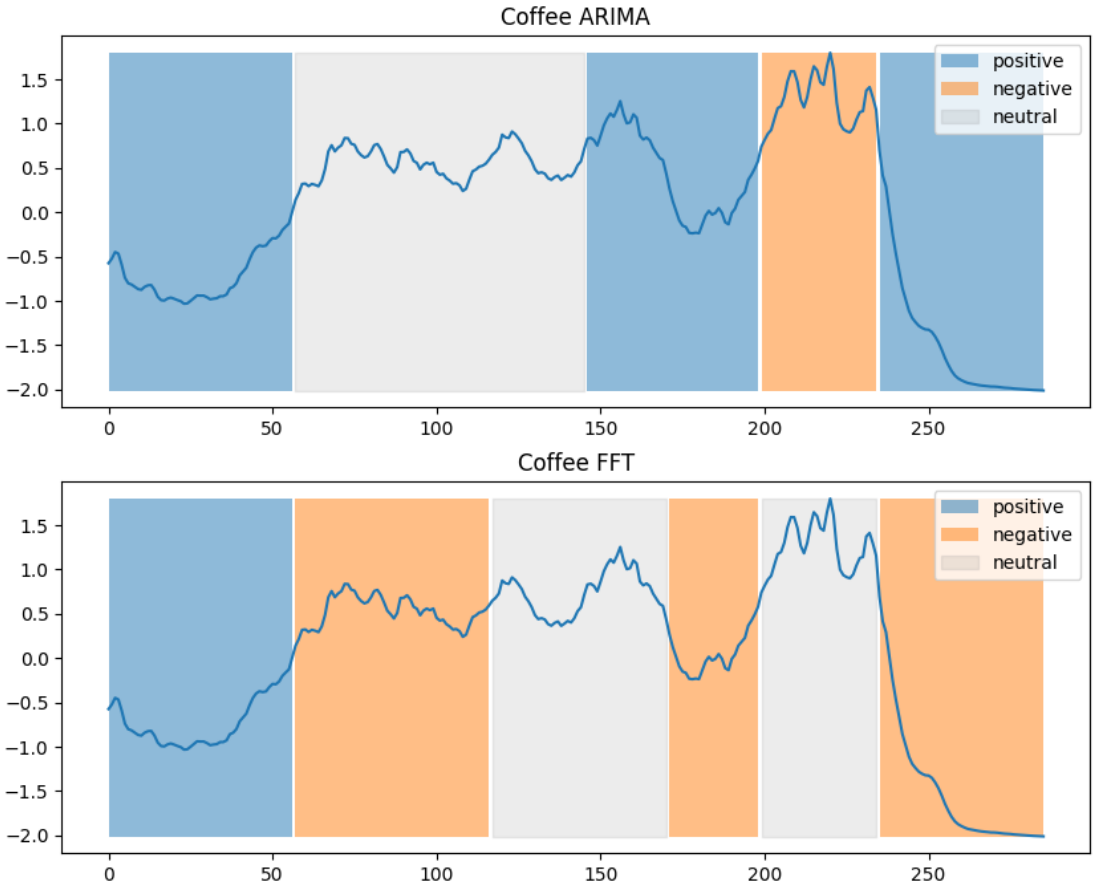


Figure 5.7 LIME explanations for the ARIMA and FFT models on the Coffee dataset. Positive presents evidence for the predicted class.

5.3 LIME method

The LIME method explains the given data point by highlighting essential parts of it. In Figure 5.7, you can see an example of such visualization. The method highlights supporting and countering forces, representing positive and negative contributions to the classification. That means it tells what parts of the time series are essential for the classifications. Additionally, it gives information about unimportant parts of the time series.

It is interesting to see that the last stripe is the same for both models, except inverted, which means that the models may behave arbitrarily.

The expectation was that the peaks of the time series would be more important, but as shown in the case of FFT, it is not the case and small perturbations in that part of the time series had no significant effect on the classification.

In both cases start of the time series seems relevant to the classification. It could be the case that for the ARIMA model, these time series parts are highly influencing the the overall trajectory of the classification because initial values from which the time series is modeled in the case of ARIMA present in that part.

Figure 5.8 contains the same two graphs, but for the case of the ECG200 dataset, which should be much more interpretable than coffee spectrographs. In this case, there is a much larger agreement between the models. Both models predicted the same class, so it seems like there is

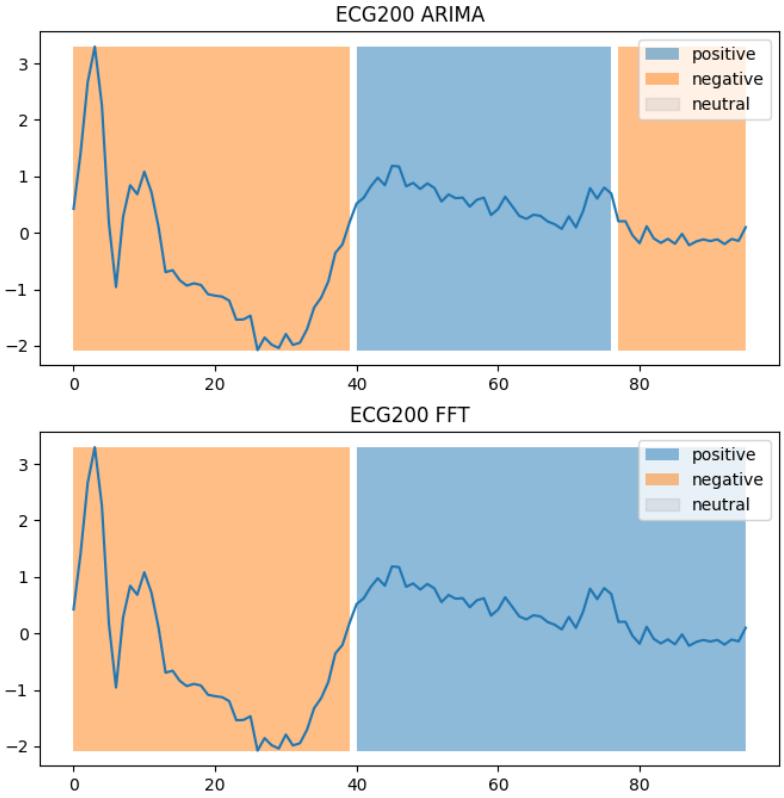


Figure 5.8 LIME explanations for the ARIMA and FFT models on the ECG200 dataset. Positive presents evidence for the predicted class.

something important in that part. They predicted the correct class with a 0.94 probability in the case of the ARIMA model and a 0.98 probability in the case of the FFT model.

The given method is also used to explain the ROCKET classifier, which uses randomly generated convolutional kernels to make the predictions. An example of that is available in Figure 5.9. There, it seems that the last part is not important for the prediction power of the classifier. It looks like it positively contributes to class 0 of the classification problem, but that class has almost zero probability of being predicted. The rest of the explanation is used positively to predict the class with has almost certain probability of being correct.

That part of the signal contains the impulse from the patient's hearth recorded using the ECG machine. The accuracy on the test dataset for the complete time series is 0.91 using the rocket classifier with the logistic regression. After the deletion of the last 17 data points from the time series, the accuracy score of the ECG200 dataset has risen to 0.92. The non-formal hypothesis presented in this paragraph is highly probable.

To further evaluate the explainability of different methods on classification problems, it is vital to have a good understating of the problem domain. In the case of ECG datasets, it is simpler to see the importance of segments in the series.

5.3.0.1 LIME on ArrowHead dataset

The ArrowHead dataset is interesting because it is an image outline time series. That means parts of the series should be highly influential in the classification process. Additionally, those parts of the series should be easily interpretable, making it easier to see if the classifier learned from the relevant part.

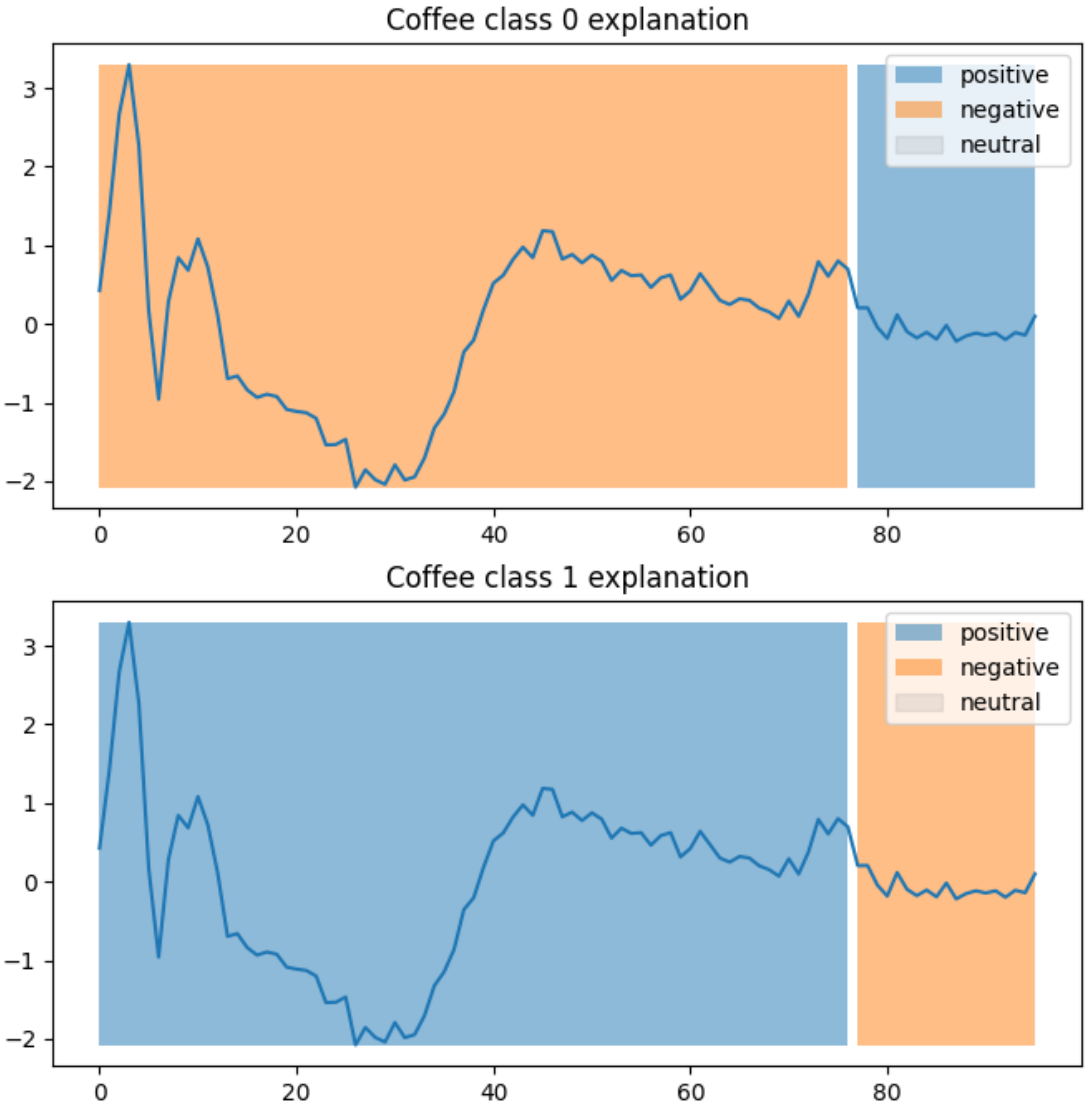


Figure 5.9 LIME explanations for the ROCKET classifier

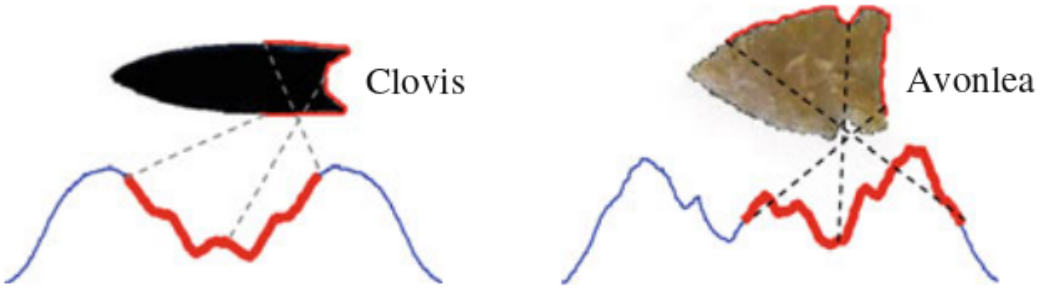


Figure 5.10 ArrowHead image to time series conversion. Shows the important part of the data set, which should also be important for the classifier. The image is taken from Ye et al. [6]

Figure 5.10 shows the conversion from the image outline. It is done by adjusting the images approximately to the same size. Then convert the time series using the angle-based method [55], and then concatenate the time series to itself to make it rotation invariant.

In Figure 5.11 it is possible to see the important parts of the time series data for some local classification. This concrete classification resulted in the following probabilities $0.708, 0.008, 0.282$ for Avonlea, Clovis, and Mixed arrowheads, respectively. Graphs showing Avonlea and The mix is just inverted, which makes sense when noticing that a mixed one would be closer to Avonlea than a completely different shape. Crucially, it seems to be the case that for both Avonlea and Mix datasets, the middle part plays a role. In Clovis's case, that part is deemed neutral, which means the classifier did not use that part for guessing Clovis.

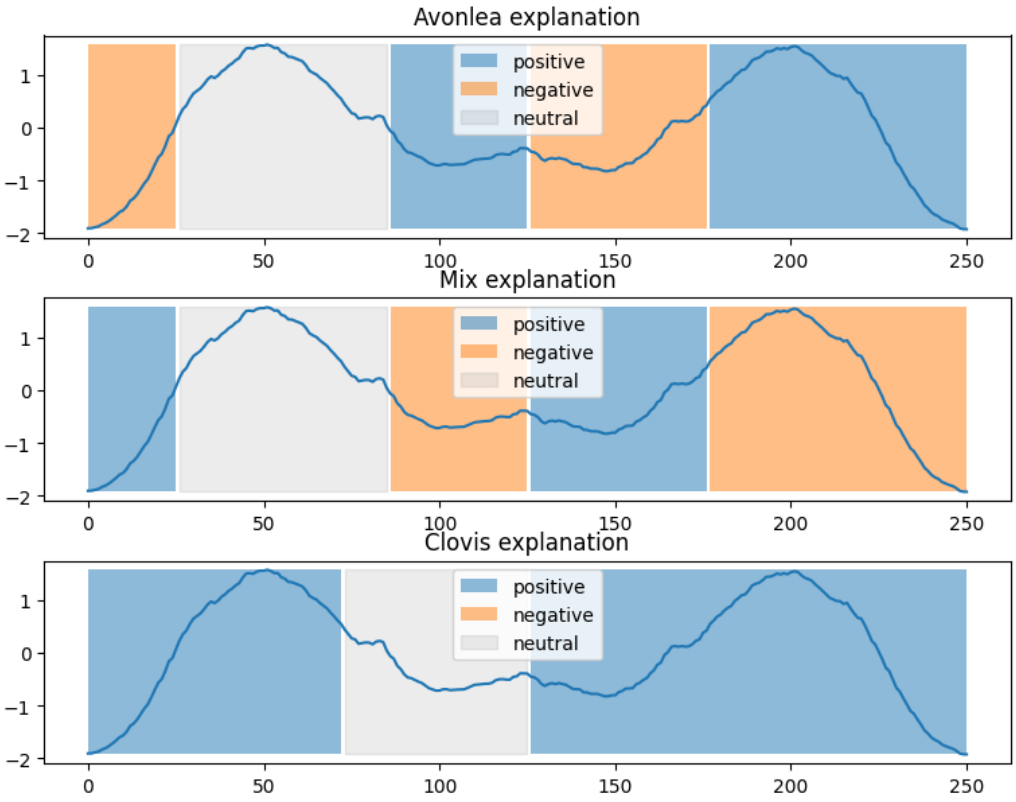


Figure 5.11 Lime explanation of the ROCKET classifier on ArrowHead dataset

Chapter 6
Conclusion

The thesis covered an important and neglected part of the time series classification: whether the classifier can generalize essential parts of the time series to form the prediction. The importance comes from the overwhelming usage of time series in medicine, the economy, and other crucial parts of our society. Not being able to understand the artificial thought process of these models can be detrimental to both individuals and society.

In the first chapter, the state-of-the-art methods and relevant literature for both the classification and explainability problems of time series are introduced to the reader. Various types of methods for classification were researched from the symbolic classifiers leveraging Fourier transform and distance-based method using elastic distance measures like DTW to feature selection using the ROCKET classifier and convolutional kernels.

In the case of explainability, both white-box and black-box methods were studied. White-box models are groups of explanators that leverage the structure of the classifier or are inherently explainable. Black box models are structure independent and functional for explaining any model. The most notable examples of black box methods include LIME and SHAP.

After the literature review, this thesis proposes one classification method to apply black box methods based on autoregressive moving average, ARIMA models, and a modification of the LIME method, which initially contained only local surrogate explanation models for tabular, text, and image data. That was achieved by leveraging that image and time series are topologically equal, where the time series is just an image represented as a row of 1 channel pixels. The LIME method was applied to the ARIMA method proposed and additionally to the state-of-the-art classifier called ROCKET.

The last chapter considers some experiments in the performance and explainability departments for the proposed methods. It firstly measures accuracy scores over the wide variety of publicly available time series classification datasets. The time series datasets used in the experiment are from the UCR archive, which is already train/test spitted to remove the possible bias from the improper data splitting. After that, it shows some visualizations used to enhance the interpretability of the models and discusses the possible interpretations. In the end, it states that domain knowledge of the data presented is needed for one to appreciate the explanations. It is vital to consult domain experts about the validity of the models.

There is still much room for additional research. It would be interesting to develop LIME time series local surrogate model that is different from the image one and compare them to each other. Additionally more formal hypothesis testing of both classification and explainability methods is welcomed, maybe by reaching out and surveying the domain experts or backing up claims about the importance of different segments in time series using more rigorous statistical apparatus.

Ultimately, it is important to prioritize explainability in the field of machine learning. Even

though it is not that profitable of a sector as recommendation systems, market or text prediction is the most important one if humanity wants to deploy powerful models in a responsible manner.

Bibliography

1. MIDDLEHURST, Matthew; LARGE, James; FLYNN, Michael; LINES, Jason; BOSTROM, Aaron; BAGNALL, Anthony. *HIVE-COTE 2.0: A New Meta Ensemble for Time Series Classification* [online]. 2021-04-15 [visited on 2023-04-14]. Available from DOI: 10.48550/arXiv.2104.07551.
2. BELLMAN, R.; KALABA, R. On Adaptive Control Processes. *IRE Transactions on Automatic Control*. 1959, vol. 4, no. 2, pp. 1–9. ISSN 1558-3651. Available from DOI: 10.1109/TAC.1959.1104847.
3. COOLEY, J.; LEWIS, P.; WELCH, P. The Finite Fourier Transform. *IEEE Transactions on Audio and Electroacoustics*. 1969, vol. 17, no. 2, pp. 77–85. ISSN 1558-2582. Available from DOI: 10.1109/TAU.1969.1162036.
4. LUBBA, Carl H.; SETHI, Sarab S.; KNAUTE, Philip; SCHULTZ, Simon R.; FULCHER, Ben D.; JONES, Nick S. Catch22: CAnonical Time-series CHaracteristics. *Data Mining and Knowledge Discovery* [online]. 2019, vol. 33, no. 6, pp. 1821–1852 [visited on 2023-03-04]. ISSN 1573-756X. Available from DOI: 10.1007/s10618-019-00647-x.
5. BREIMAN, Leo. Random Forests. *Machine Learning* [online]. 2001, vol. 45, no. 1, pp. 5–32 [visited on 2023-03-05]. ISSN 1573-0565. Available from DOI: 10.1023/A:1010933404324.
6. YE, Lexiang; KEOGH, Eamonn. Time Series Shapelets: A Novel Technique That Allows Accurate, Interpretable and Fast Classification. *Data Mining and Knowledge Discovery* [online]. 2011, vol. 22, no. 1, pp. 149–182 [visited on 2023-03-04]. ISSN 1573-756X. Available from DOI: 10.1007/s10618-010-0179-5.
7. SCHÄFER, Patrick. The BOSS Is Concerned with Time Series Classification in the Presence of Noise. *Data Mining and Knowledge Discovery* [online]. 2015, vol. 29, no. 6, pp. 1505–1530 [visited on 2023-03-04]. ISSN 1573-756X. Available from DOI: 10.1007/s10618-014-0377-7.
8. DEMPSTER, Angus; PETITJEAN, François; WEBB, Geoffrey I. ROCKET: Exceptionally Fast and Accurate Time Series Classification Using Random Convolutional Kernels. *Data Mining and Knowledge Discovery* [online]. 2020, vol. 34, no. 5, pp. 1454–1495 [visited on 2023-04-14]. ISSN 1384-5810, ISSN 1573-756X. Available from DOI: 10.1007/s10618-020-00701-z.
9. MOCHAOURAB, Rami; VENKITARAMAN, Arun; SAMSTEN, Isak; PAPAPETROU, Panagiotis; ROJAS, Cristian R. Post Hoc Explainability for Time Series Classification: Toward a Signal Processing Perspective. *IEEE Signal Processing Magazine*. 2022, vol. 39, no. 4, pp. 119–129. ISSN 1558-0792. Available from DOI: 10.1109/MSP.2022.3155955.
10. RIBEIRO, Marco; SINGH, Sameer; GUESTRIN, Carlos. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In: 2016, pp. 97–101. Available from DOI: 10.18653/v1/N16-3020.

11. ŠTRUMBELJ, Erik; KONONENKO, Igor. Explaining Prediction Models and Individual Predictions with Feature Contributions. *Knowledge and Information Systems* [online]. 2014, vol. 41, no. 3, pp. 647–665 [visited on 2023-03-06]. ISSN 0219-3116. Available from DOI: 10.1007/s10115-013-0679-x.

12. SUNDARARAJAN, Mukund; NAJMI, Amir. *The Many Shapley Values for Model Explanation* [online]. 2020-02-07 [visited on 2023-03-06]. Available from DOI: 10.48550/arXiv.1908.08474.

13. PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-Learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2011, vol. 12, pp. 2825–2830.

14. BAGNALL, Anthony; BOSTROM, Aaron; LARGE, James; LINES, Jason. *The Great Time Series Classification Bake Off: An Experimental Evaluation of Recently Proposed Algorithms. Extended Version* [online]. 2016-02-04 [visited on 2023-04-28]. Available from DOI: 10.48550/arXiv.1602.01711.

15. BAGNALL, Anthony; LINES, Jason; BOSTROM, Aaron; LARGE, James; KEOGH, Eamonn. The Great Time Series Classification Bake off: A Review and Experimental Evaluation of Recent Algorithmic Advances. *Data Mining and Knowledge Discovery* [online]. 2017, vol. 31, no. 3, pp. 606–660 [visited on 2023-02-04]. ISSN 1573-756X. Available from DOI: 10.1007/s10618-016-0483-9.

16. HYNDMAN, Rob J.; ATHANASOPOULOS, George. 6.1 Time Series Components. In: *Forecasting: Principles and Practice (2nd Ed)* [online]. [N.d.] [visited on 2023-05-01]. Available from: https://otexts.com/fpp2/components.html.

17. HYNDMAN, Rob J.; ATHANASOPOULOS, George. 2.3 Time Series Patterns. In: *Forecasting: Principles and Practice (2nd Ed)* [online]. [N.d.] [visited on 2023-05-01]. Available from: https://otexts.com/fpp2/tspatterns.html#tspatterns.

18. ROBERTO, Heredia. *English: Residential Seasonal Decompose* [online]. 2021 [visited on 2023-03-29]. Available from: https://commons.wikimedia.org/wiki/File:Residential_seasonal_decompose.jpg#filelinks.

19. HYNDMAN, Rob J.; ATHANASOPOULOS, George. 8.1 Stationarity and Differencing. In: *Forecasting: Principles and Practice (2nd Ed)* [online]. [N.d.] [visited on 2023-05-01]. Available from: https://otexts.com/fpp2/stationarity.html.

20. MAHARAJ, Elizabeth Ann; D'URSO, Pierpaolo; CAIADO, Jorge. 3.2 Distance Measures. In: *Time Series Clustering and Classification*. New York: Chapman and Hall/CRC, 2019, p. 30. ISBN 978-0-429-05826-4. Available from DOI: 10.1201/9780429058264.

21. MAHARAJ, Elizabeth Ann; D'URSO, Pierpaolo; CAIADO, Jorge. Dynamic Time Warping. In: *Time Series Clustering and Classification*. New York: Chapman and Hall/CRC, 2019, pp. 53, 54. ISBN 978-0-429-05826-4. Available from DOI: 10.1201/9780429058264.

22. ZHANG, Zheng; TAVENARD, Romain; BAILLY, Adeline; TANG, Xiaotong; TANG, Ping; CORPETTI, Thomas. Dynamic Time Warping under Limited Warping Path Length. *Information Sciences* [online]. 2017, vol. 393, pp. 91–107 [visited on 2023-05-01]. ISSN 0020-0255. Available from DOI: 10.1016/j.ins.2017.02.018.

23. QUINLAN, J. R. Induction of Decision Trees. *Machine Learning* [online]. 1986, vol. 1, no. 1, pp. 81–106 [visited on 2023-03-30]. ISSN 1573-0565. Available from DOI: 10.1007/BF00116251.

24. FRIEDMAN, Jerome. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics*. 2001, vol. 29, pp. 1189–1232. Available from DOI: 10.2307/2699986.

25. GRAPS, Amara. An Introduction to Wavelets. [N.d.].

26. AGATELLER, Created by. *Schematic Diagram of Normal Sinus Rhythm for a Human Heart as Seen on ECG (with English Labels).* [Online]. 13 January 2007, 23:40 [visited on 2023-03-04]. Available from: `https://commons.wikimedia.org/wiki/File:SinusRhythmLabels.svg`.

27. DENG, Houtao; RUNGER, George; TUV, Eugene; VLADIMIR, Martyanov. A Time Series Forest for Classification and Feature Extraction. *Information Sciences* [online]. 2013, vol. 239, pp. 142–153 [visited on 2023-03-04]. ISSN 0020-0255. Available from DOI: `10.1016/j.ins.2013.02.030`.

28. TAVENARD, Romain; FAOUZI, Johann; VANDEWIELE, Gilles; DIVO, Felix; ANDROZ, Guillaume; HOLTZ, Chester; PAYNE, Marie; YURCHAK, Roman. *Aligning Discovered Shapelets with Timeseries — Tslearn 0.5.3.2 Documentation* [online] [visited on 2023-04-02]. Available from: `https://tslearn.readthedocs.io/en/stable/auto_examples/classification/plot_shapelet_locations.html#sphx-glr-auto-examples-classification-plot-shapelet-locations-py`.

29. TAVENARD, Romain; FAOUZI, Johann; VANDEWIELE, Gilles; DIVO, Felix; ANDROZ, Guillaume; HOLTZ, Chester; PAYNE, Marie; YURCHAK, Roman; RUSSWURM, Marc; KOLAR, Kushal; WOODS, Eli. Tslearn, a Machine Learning Toolkit for Time Series Data. *The Journal of Machine Learning Research*. 2020, vol. 21, no. 1, 118:4686–118:4691. ISSN 1532-4435.

30. GRABOCKA, Josif; SCHILLING, Nicolas; WISTUBA, Martin; SCHMIDT-THIEME, Lars. Learning Time-Series Shapelets. In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* [online]. New York, NY, USA: Association for Computing Machinery, 2014, pp. 392–401 [visited on 2023-03-04]. KDD '14. ISBN 978-1-4503-2956-9. Available from DOI: `10.1145/2623330.2623613`.

31. SCHÄFER, Patrick; HÖGQVIST, Mikael. SFA: A Symbolic Fourier Approximation and Index for Similarity Search in High Dimensional Datasets. In: *Proceedings of the 15th International Conference on Extending Database Technology* [online]. New York, NY, USA: Association for Computing Machinery, 2012, pp. 516–527 [visited on 2023-04-02]. EDBT '12. ISBN 978-1-4503-0790-1. Available from DOI: `10.1145/2247596.2247656`.

32. PEARSON, Karl. LIII. On Lines and Planes of Closest Fit to Systems of Points in Space [online]. 1901 [visited on 2023-04-20]. Available from DOI: `10.1080/14786440109462720`.

33. *Encyclopedia Titanica* [online] [visited on 2023-04-08]. Available from: `https://www.encyclopedia-titanica.org/`.

34. LAUGEL, Thibault; LESOT, Marie-Jeanne; MARSALA, Christophe; RENARD, Xavier; DETYNIECKI, Marcin. Comparison-Based Inverse Classification for Interpretability in Machine Learning. In: MEDINA, Jesús; OJEDA-ACIEGO, Manuel; VERDEGAY, José Luis; PELTA, David A.; CABRERA, Inma P.; BOUCHON-MEUNIER, Bernadette; YAGER, Ronald R. (eds.). *Information Processing and Management of Uncertainty in Knowledge-Based Systems. Theory and Foundations*. Cham: Springer International Publishing, 2018, pp. 100–111. Communications in Computer and Information Science. ISBN 978-3-319-91473-2. Available from DOI: `10.1007/978-3-319-91473-2_9`.

35. HARMAN, Radoslav; LACKO, Vladimír. On Decompositional Algorithms for Uniform Sampling from N-Spheres and n-Balls. *Journal of Multivariate Analysis* [online]. 2010, vol. 101, no. 10, pp. 2297–2304 [visited on 2023-04-16]. ISSN 0047-259X. Available from DOI: `10.1016/j.jmva.2010.06.002`.

36. BECKER, Barry. *Adult* [online]. UCI Machine Learning Repository, 1996 [visited on 2023-04-08]. Available from DOI: `10.24432/C5XW20`.

37. SHAPLEY, Lloyd S. *A Value for N-Person Games* [online]. 1952-03-18 [visited on 2023-03-06]. RAND Corporation. Available from: https://www.rand.org/pubs/papers/P295.html.

38. LUNDBERG, Scott M.; LEE, Su-In. A Unified Approach to Interpreting Model Predictions. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2017, pp. 4768–4777. NIPS'17. ISBN 978-1-5108-6096-4.

39. ERDEM, Ahmet. *Aerdem4/Lofo-Importance* [online]. 2023 [visited on 2023-04-10]. Available from: https://github.com/aerdem4/lofo-importance.

40. JASTRZEBSKA, Agnieszka; HOMENDA, Wladyslaw; PEDRYCZ, Witold. ARIMA Feature-Based Approach to Time Series Classification. In: GROEN, Derek; de MULATIER, Clélia; PASZYNSKI, Maciej; KRZHIZHANOVSKAYA, Valeria V.; DONGARRA, Jack J.; SLOOT, Peter M. A. (eds.). *Computational Science – ICCS 2022*. Cham: Springer International Publishing, 2022, pp. 192–199. Lecture Notes in Computer Science. ISBN 978-3-031-08754-7. Available from DOI: 10.1007/978-3-031-08754-7_26.

41. HYNDMAN, Rob J.; ATHANASOPOULOS, George. 9. ARIMA Models. In: *Forecasting: Principles and Practice (3rd Ed)* [online]. [N.d.] [visited on 2023-04-07]. Available from: https://otexts.com/fpp3/arima.html.

42. WANG, Xiaozhe; SMITH, Kate; HYNDMAN, Rob. Characteristic-Based Clustering for Time Series Data. *Data Mining and Knowledge Discovery* [online]. 2006, vol. 13, no. 3, pp. 335–364 [visited on 2023-03-07]. ISSN 1573-756X. Available from DOI: 10.1007/s10618-005-0039-x.

43. HYNDMAN, Rob J.; KHANDAKAR, Yeasmin. Automatic Time Series Forecasting: The Forecast Package for R. *Journal of Statistical Software* [online]. 2008, vol. 27, pp. 1–22 [visited on 2023-03-07]. ISSN 1548-7660. Available from DOI: 10.18637/jss.v027.i03.

44. SEABOLD, Skipper; PERKTOLD, Josef. Statsmodels: Econometric and Statistical Modeling with Python. In: [online]. Austin, Texas, 2010, pp. 92–96 [visited on 2023-04-26]. Available from DOI: 10.25080/Majora-92bf1922-011.

45. GARZA, Federico; CANSECO, Max Mergenthaler; CHALLÚ, Cristian; G. OLIVARES, Kin. *StatsForecast: Lightning Fast Forecasting with Statistical and Econometric Models* [PyCon Salt Lake City, Utah, US 2022]. 2022. Available also from: https://github.com/Nixtla/statsforecast.

46. FAOUZI, Johann; JANATI, Hicham. Pyts: A Python Package for Time Series Classification. *Journal of Machine Learning Research*. 2020, vol. 21, no. 46, pp. 1–6. Available also from: http://jmlr.org/papers/v21/19-763.html.

47. PROKHORENKOVA, Liudmila; GUSEV, Gleb; VOROBEV, Aleksandr; DOROGUSH, Anna Veronika; GULIN, Andrey. *CatBoost: Unbiased Boosting with Categorical Features* [online]. 2019-01-20 [visited on 2023-04-29]. Available from DOI: 10.48550/arXiv.1706.09516.

48. CHRIST, Maximilian; BRAUN, Nils; NEUFFER, Julius; KEMPA-LIEHR, Andreas W. Time Series FeatuRe Extraction on Basis of Scalable Hypothesis Tests (Tsfresh – A Python Package). *Neurocomputing* [online]. 2018, vol. 307, pp. 72–77 [visited on 2023-04-29]. ISSN 0925-2312. Available from DOI: 10.1016/j.neucom.2018.03.067.

49. LÖNING, Markus; BAGNALL, Anthony; GANESH, Sajaysurya; KAZAKOV, Viktor; LINES, Jason; KIRÁLY, Franz J. *Sktime: A Unified Interface for Machine Learning with Time Series* [online]. 2019-09-17 [visited on 2023-04-16]. Available from DOI: 10.48550/arXiv.1909.07872.

50. KALPAKIS, K.; GADA, D.; PUTTAGUNTA, V. Distance Measures for Effective Clustering of ARIMA Time-Series. In: *Proceedings 2001 IEEE International Conference on Data Mining*. 2001, pp. 273–280. Available from DOI: 10.1109/ICDM.2001.989529.

51. RIBEIRO, Marco Tulio Correia. *Lime* [online]. 2023 [visited on 2023-04-13]. Available from: https://github.com/marcotcr/lime.

52. BRIANDET, R.; KEMSLEY, E. K.; WILSON, R. H. Discrimination of Arabica and Robusta in Instant Coffee by Fourier Transform Infrared Spectroscopy and Chemometrics. *Journal of agricultural and food chemistry* [online]. 1996, vol. 44, no. 1, pp. 170–174 [visited on 2023-03-28]. ISSN 1520-5118. Available from DOI: 10.1021/jf950305a.

53. AL-JOWDER, Osama; KEMSLEY, E. K.; WILSON, Reginald H. Detection of Adulteration in Cooked Meat Products by Mid-Infrared Spectroscopy. *Journal of Agricultural and Food Chemistry*. 2002, vol. 50, no. 6, pp. 1325–1329. ISSN 0021-8561. Available from DOI: 10.1021/jf0108967.

54. OLSZEWSKI, Robert Thomas. *Generalized Feature Extraction for Structural Pattern Recognition in Time-Series Data*. USA, 2001. PhD thesis. Carnegie Mellon University.

55. KEOGH, Eamonn; WEI, Li; XI, Xiaopeng; LEE, Sang-Hee; VLACHOS, Michail. LB_Keogh Supports Exact Indexing of Shapes under Rotation Invariance with Arbitrary Representations and Distance Measures. In: *Proceedings of the 32nd International Conference on Very Large Data Bases*. Seoul, Korea: VLDB Endowment, 2006, pp. 882–893. VLDB '06.

Content of the annexed ZIP file

- `impl` ..... implementation source codes
- `thesis` ..... thesis source code with assets in LaTeX format
- `text` ..... rendered text of the thesis
  - `thesis.pdf` ..... the thesis in the PDF format