Bachelorarbeit

Classification of Road Conditions and Types Based on Mobile Devices Telemetry Data

Verfasser

Lorenz Kummer

angestrebter akademischer Grad

Bachelor of Science (BSc)

Wien, 2020

Studienkennzahl lt. Studienblatt:	A 033 521
Fachrichtung:	Informatik - Allgemein
Betreuerin / Betreuer:	Univ.-Prof. Dipl.-Inform.Univ. Dr. Claudia Plant Dipl.-Ing. Lukas Miklautz, BSc

Contents

Abstract

While efficient and accurate solutions for detecting known patterns in time series exist for at least for two decades [1, 9, 21], the task of labeling real world data as required by applications found in insurance companies [27, 30] still leaves many potentially promising options unexplored [60]. Based on the Sussex-Huawei Locomotion and Transportation Dataset [24], this work compares the quality of statistically extracted features [11, 10], grammar [32, 19] and similarity based [35, 65, 63, 16, 23, 64] motif discovery with one another as well as the results found during the Sussex-Huawei Locomotion Challenge [29]. In a machine learning approach carefully engineered based on existing solutions to similar problems [36, 2], we were able show that the classification quality of the features extracted by the motif based approach are at least equal to the statistical approach and more robust to the change point detection problem [4], achieving a classification rate of up to 85.2% in a binary road type classification problem (City, Countryside) without change point detection, while the features engineered by the statistical approach only resulted in a classification accuracy of 79.0% without and 85.2% with simulated change point detection. In a first step, the data is preprocessed and normalized, then the features are extracted and backwards selected using wrapper or filter methods [43, 55]. Finally, the classification quality if evaluated for a number of different hyper-parameter optimized classifiers (Random Forests, CART-Trees, SVM).

1 Introduction

1.1 Motivation and Relevancy

The real world applications for automatically discovering whether a vehicle is travelling in urban or rural conditions are manifold. One obvious application is risk management by insurance companies offering, for example, liability or more comprehensive insurance services to drivers and drivers who are engaging in dangerous driving style can be made aware of their behaviour and other drivers can be advised to take caution as well [27, 60, 3].

The insurance domain specific task of classifying weather certain drivers behaviour is risky can be simplified by classifying the road type on which a vehicle is traveling, given that the risk of an accident as well as its gravity correlate with the type of road a vehicle is traveling on [5, 17]. Husnjak et al. argued in 2018 as well that road type is a relevant parameter for premium calculation and billing process in motor insurance [28].

1.2 Accelerometer as Road Type Indicator

Technically, one possible solution to achieve this classification is to classify the road type a vehicle is travelling using accelerometer data. This solution is based on assumptions that include without limitation the following:

- Different road types typically have different speed limits $V_{max} = \{v_{country}, v_{city}, v_{highway}\}$. A vehicle accelerating with $a_x(t) = \frac{\delta v_x}{\delta t} = const$ will therefore require different acceleration periods to reach the speed limit from a stand still and to come to a stand still when traveling at the speed limit (note though that this assumption might not hold under non-linear acceleration). These variations in deceleration and acceleration measurements can be used to detect traffic conditions characteristic for certain road types, such as stop-and-go traffic which is most common on urban conditions, and thus be used as an indicator for the road type a vehicle is traveling on.

- Road surfaces are of varying quality, depending on the purpose of the road. Highways characteristically have smooth, high quality surfaces and few potholes as they are designed for high speed traveling, while the same is not true, for example, for dirt roads. Given that every pothole a vehicle is travelling over produces a characteristic high frequency peak in $a_z(t)$, it can be assumed that various road types will produce these signal at various frequencies.

- Cornering is an activity that changes the direction a vehicle is traveling and will thus produce variations that affect $a_y(t)$. Different road types require different cornering techniques; highways typically have few long, smooth high speed bends, country roads often have tighter bends that are driven through at medium speeds while cornering in urban conditions usually happens at slow speeds but at higher frequency.

2 Related Work

Numerous papers exists regarding the classification of driving behaviour using accelerometer data. For brevity, only the three most contextually relevant are explicitly discussed below. A more general overview is provided by the following survey: [60]

2.1 Driver Risk Evaluation

Joubert et al. introduce a method that rates individual driver behaviour relative to others by discretising the data into a tractable and finite risk space instead of trying to extract harsh events from large volumes of data. While this is a very flexible approach that allows to classify driving style into acceptable and unacceptable behaviour relative to a sample population depending on risk appetite, no classification regarding traffic or road condition is made.

Also, Jourbert et al. did not use data from smartphone based accelerometers but from devices that were installed by insurance companies in the context of pay-as-you-drive policies in fixed positions in the vehicles participating in the study. Using accelerometers fixed to the vehicles gave them access to high quality data which likely suffered much less from the distortions commonly found in smartphone accelerometer data [30].

2.2 Road Quality Classification

Allouch et al. created a real-time smartphone app called RoadSense that, using smartphone based accelerometer data, automatically predicts the quality of the road based on a tri-axial accelerometer and a gyroscope. They addressed the problem of the distortions contained in smartphone accelerometer data in the preprocessing step by first applying a low pass filter and then combining the accelerometer readings three components into a single magnitude score using the euclidean norm. For classification, they tested a C4.5 decision tree, a SVM and a Naive Bayes classifier, of which the C4.5 showed to work best for the problem at hand [2].

2.3 Driver Behaviour and Traffic Events

Predic et al. used smartphone accelerometer data to classify types of driver behaviour and types of general traffic events. During preprocessing, they addressed the problem of the distortions in smartphone accelerometer data by extracting the gravity force vector and using it to reconstruct the device orientation and then applied a low pass filter to remove outliers. Feature engineering and classification is done using DFT and a cross correlation based technique. Predic et al found that the most distinctive events in their sample were dynamic traffic events (near-incidents), sudden braking and acceleration, obstacle avoidance, lateral skidding and abrupt lane changes while the most distinctive traffic characteristics were congestion and average speeds and road surface conditions and quality [53].

3 Problem Setting

The problem setting is defined by the given dataset, the feature extraction and selection as well as the classification algorithms chosen.

3.1 Dataset

The Sussex-Huawei Locomotion and Transportation Dataset was created as a publicly available benchmark data set Gjorski et al. in 2018:
"The dataset comprises 7 months of measurements, collected from all sensors of 4 smartphones carried at typical body locations, including the images of a body-worn camera, while 3 participants used 8 different modes of transportation in the south- east of the United Kingdom, including in London [Fig 1]. In total 28 context labels were annotated, including transportation mode, participant's posture, inside/outside location, road conditions, traffic conditions, presence in tunnels, social interactions, and having meals. The total amount of collected data exceed 950 GB of sensor data, which corresponds to 2812 hours of labelled data and 17562 km of traveled distance."[24]

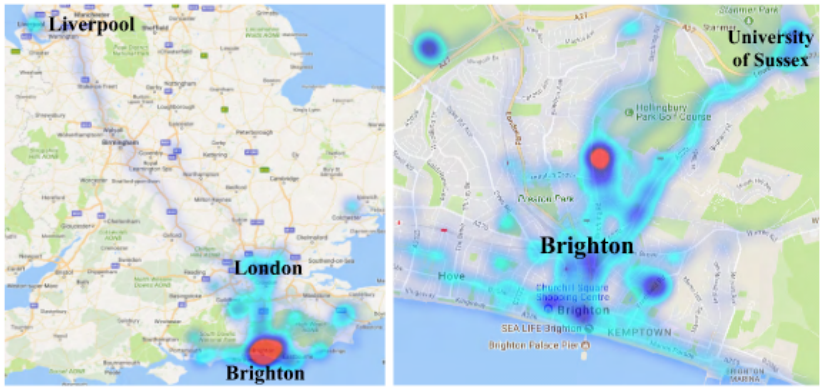


Figure 1: Sussex-Huawei Dataset Recording Locations Heat Map [24]

This makes the Sussex-Huawei Locomotion and Transportation Dataset a very comprehensive high quality mobile devices telemetry dataset (summary in Table 1) ideal for the development of our application.

Carriers	Sensors	Modes	Labels	Size	Length	Distance
3	16	8	28	950 GB	2812 hours	17562 km

Table 1: Sussex-Huawei Dataset Overview

The sensor measurements listed in Table 4 were recorded at the highest available sampling rate offered by the Android system and annotated by the

participants using a purpose-built Android app with two types of context labels, one for the activity of the recording participant (Table 2), one for the road type (Table 3) [24]. The distribution of the labels in the dataset is shown in Fig. 2.

For the experiments in this work, trips that were labelled as activity "Car", and road type "City" and "Country Road" were used.

Activity
Still: standing or sitting; inside or outside
Walking: inside or outside
Run
Bike (Bicycle)
Bus: standing or sitting; lower deck or upper deck
Car: as driver or as passenger
Train: standing or sitting
Subway: standing or sitting

Table 2: Sussex-Huawei Dataset Activity Labels

Road Type
City
Motorway
Countryside
Dirt road

Table 3: Sussex-Huawei Dataset Road Labels

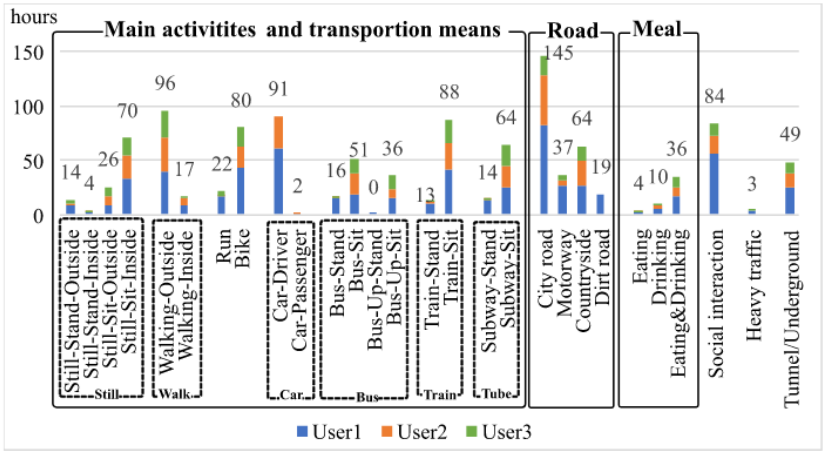


Figure 2: Sussex-Huawei Dataset Label Distribution [24]

Sensors	**Sensors**
Accelerometer	Ambient light
Gyroscope	Battery level and temperature
Magnetometer	Satellite reception
Orientation in quaternions	WiFi reception
Gravity	Mobile phone cell reception
Linear acceleration	Location obtained from satellites
Ambient pressure	Audio
Google's activity recognition API	

Table 4: Sussex-Huawei Dataset Sensors

3.2 Statistical Features

Statistical features are extracted via the FeatuRe Extraction based on Scalable Hypothesis tests (FRESH) algorithm, which was first introduced by Christ et al. in 2016 [11] and is publicly available as a Python package [10] called TSFRESH. The FRESH algorithm characterizes a time series by a mapping

$$\theta_k : \mathbb{R}^{n_t^{(j)}} \to \mathbb{R}$$

which captures a specific aspect of the time series, which is called a feature. One example for such a mapping is the maximum operator

$$\theta_{max}(S_{i,j}) = max\{s_{i,j,1}, s_{i,j,2}, ..., s_{i,j,n_t^{(j)}}\}$$

which quantifies the maximal value encountered in time series $S_{i,j}$. Other examples for feature mappings θ_k of time series could be their mean, the number of peaks with a certain steepness or their periodicity [11]. Given $m \cdot n$ time series recorded from n sensors and m devices and n_f different time series feature mappings, each column of the resulting feature matrix $\mathbb{X} \in \mathbb{R}^{m \times n_\Phi}$ (with $n_\Phi = n \cdot n_f + n_i$ where n_i denotes the number of features generated from device specific meta information) comprises a vector $X_\phi \in \mathbb{R}^m$ capturing a specific feature of all considered devices m. This feature matrix $\mathbb{X}$, along with a target vector Y containing the labels, can then be used for supervised classification algorithms [11].

Notation	Explanation
$s_{i,j,t}$	Measurement of device i with sensor j at time t
$S_{i,j}$	Time series of measurements of sensor j of device i
$n_t^{(j)}$	Length of $S_{i,j}$ of sensor j of device i at frequency Δt
$\theta_k(S_{i,j}) : \mathbb{R}^{n_t^{(j)}} \to \mathbb{R}$	Mapping of type k of time series $S_{i,j}$ from $\mathbb{R}^{n_t^{(j)}}$ to $\mathbb{R}$
$\mathbb{X} \in \mathbb{R}^{m \times n_\Phi}$	Feature matrix generated by TSFRESH
n_Φ	Columns (i.e. different feature mappings) of $\mathbb{X}$
X_Φ	Single feature vector of $\mathbb{X}$
Y	Target vector for machine learning applications
H_0^Φ	Null hypothesis for feature X_Φ with regards to Y
H_1^Φ	Alternative hypothesis for feature X_Φ with regards to Y

Table 5: FRESH Notation

Because typical time series contain noise and redundancies, it is important to balance meaningful but fragile and robust but non-significant features. The FRESH algorithm selects relevant features based on the following definition: *"Definition 1 (A relevant feature). A feature X_ϕ is relevant or meaningful for the prediction of Y if and only if X_ϕ and Y are not statistically independent."*[11]. Hypothesis testing is used by the FRESH algorithm to determine the statistical dependence of each feature $X_1, ..., X_\Phi, ..., X_{n_\Phi}$ and hence its relevance for predicting Y. A singular statistical test is used for checking the hypothesis:

$$H_0^\Phi = \{X_\Phi \text{ is irrelevant for predicting } Y\}$$

$$H_1^\Phi = \{X_\Phi \text{ is relevant for predicting } Y\}$$

The ouput of each hypothesis test H_0^Φ is called p-value p_Φ
The independence tests used by FRESH are the Exact Fisher test of independence, the Kolmogorov-Smirnov test (binary feature), the Kolmogorov-Smirnov test (binary target) or the Kendal rank test, depending on the feature matrix and the target variable [11].

Since the risk of a feature X_Φ being added for which H_0^Φ has falsely been rejected (α-error) is by construction of the hypothesis tests only controlled for individual features, FRESH also controls the false discovery rate (FDR), which represents the expected proportion of erroneous rejections among all rejections [11]. FRESH deploys the Benjamini-Yekutieli procedure for controlling the FDR, also referred to as false extraction rate (FER) in [11]:

$$FER = \mathbb{E}[\frac{\text{number of irrelevant extracted features}}{\text{number of irrelevant extracted features}}]$$

Putting the components described above together, the complete FRESH algorithm is given by [11]:

1. *"Perform a set of n_Φ univariate feature mappings [...] on $m \cdot n$ different time series to create the feature vectors X_Φ with Φ = 1, ..., n_Φ[11].*

2. *For each generated feature vector X_1, ..., X_{n_Φ} perform exactly one hypothesis test for the hypothesis H_0^Φ [...]. To do so, take the corresponding feature significance test [...]. Calculate the p-values p_1, ..., p_{n_Φ} of the tests.[11]*

3. *Perform the Benjamini-Yekutieli procedure under correction for dependent hypotheses for a FDR level of q on the collected p-values p_1,... , p_{n_Φ} in order to decide which null hypothesis H_0^Φ to be rejected [...]. Only return features vectors for which the respective hypothesis H_0^Φ was rejected by the procedure."*

For a complete list of the features extracted by FRESH, please visit [12]

3.3 Motifs as Features

Motifs, first described by Keogh et al. in 2002 [35], are enumerated frequently occurring patterns in time series 3 (Fig. 3). Keogh et al. initially proposed an algorithm that discovers motifs of a given length n by first discretizing a time series using piece-wise approximate aggregation (PAA) to a series of symbols and then moving a sliding window of length n over the aggregated time series to discover similar sub-sequences using a distance function. The found sub-sequences can either be sorted by similarity or by frequency of occurrence. This early motif discovery algorithm however has three significant drawbacks [35]:

1. The length n of the motif must be known beforehand. The naive solution would be to iterate over all possible motif lengths, but this would require prohibitive amounts of computational resources.

2. Scaled motifs can not be detected. Assume a motif of length n is found, the first version would not be able to detect the same repeating sub-sequence if it was scaled to, for example $2 \cdot n$.

3. The detected motifs are an approximation. Their accuracy depends on the chosen PAA as well as a parameter θ that defines the similarity threshold.

Later motif discovery algorithms introduced by researchers Keogh and Lin [65] [64] addressed these shortcomings and are therefore the primary choice for mining motifs in time series. Two such algorithms that we found best suited for this work are discussed in details below.

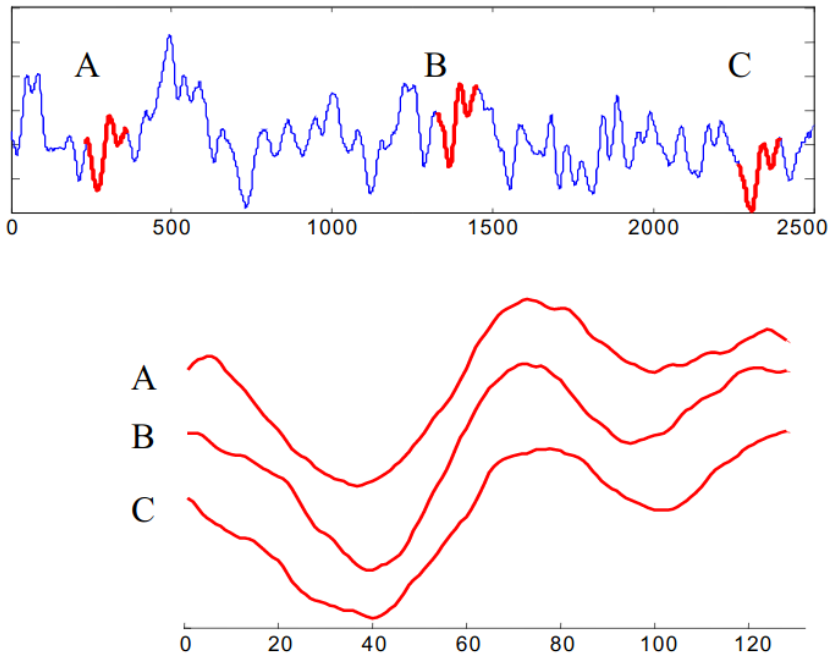


Figure 3: Example of a motif [35]

3.3.1 Motif Discovery via Matrix Profiles (MP)

A modern approach for motif discovery in time series was described in 2016 by Keogh et al. [63]. The proposed solution describes the motif discovery problem as all-pairs-similarity search characterized by the following mathematical definitions [63]:

1. *"A time series T is a sequence of real-valued numbers $t_i : T = t_1, t_2, ..., t_n$ where n is the length of T.*

2. *A sub-sequence $T_{i,m}$ of a T is a continuous subset of the values from T of length m starting from position i. $T_{i,m} = t_i, ti + 1, ..., ti + m - 1$, where $1 \leq i \leq n - m + 1$*

3. *A distance profile D is a vector of the Euclidean distances between a given query and each sub-sequence in an all-sub-sequences set (see 4).*

4. *An all-sub-sequences set A of a time series T is an ordered set of all possible sub-sequences of T obtained by sliding a window of length m across $T : A = \{T_{1,m}, T_{2,m}, ..., T_{n-m+1,m}\}$, where m is a user-defined sub-sequence length. We use $A[i]$ to denote $T_{i,m}$.*

5. *$1NN$-join function: given two all-sub-sequences sets A and B and two sub-sequences $A[i]$ and $B[j]$, a $1NN$-join function $\theta_{1nn}(A[i], B[j])$ is a*

Boolean function which returns "true" only if $B[j]$ is the nearest neighbor of $A[i]$ in the set B.

6. *Similarity join set: given all-sub-sequences sets A and B, a similarity join set J_{AB} of A and B is a set containing pairs of each sub-sequence in A with its nearest neighbor in B: $J_{AB} = \{(A[i], B[j]) : \theta_{1nn}(A[i], B[j])\}$. We denote this formally as $J_{AB} = A \bowtie_{\theta_{1nn}} B$*

7. *A matrix profile (or just profile) P_{AB} is a vector of the Euclidean distances between each pair in J_{AB} [Fig. 4].*

8. *A self-similarity join set J_{AA} is a result of similarity join of the set A with itself. We denote this formally as $J_{AB} = A \bowtie_{\theta_{1nn}} A$ We denote the corresponding matrix profile or self-similarity join profile as P_{AA}.*

9. *A matrix profile index I_{AB} of a similarity join set J_{AB} is a vector of integers where $I_{AB}[i] = j$ if $\{A[i], B[j]\} \in J_{AB}$."*

Notation	**Explanation**
T	Time series of length n
$T_{i,m}$	Sub-sequence of T of length m
D	Distance profile containing all distances between $T_{i,m} \in T$
A	All-sub-sequences-subset containing all possible $T_{i,m}$
$1NN$-join function	Similarity join function
J_{AB}	Similarity join set
P_{AB}	Matrix profile vector
J_{AA}	Self-similarity join set
I_{AB}	Matrix profile index

Table 6: MP Notation

Summarizing Keogh's approach using above definitions, for a given time series T consisting of sub-sequences $T_{i,m}$ a sliding window of length m is moved across T to obtain the all-sub-sequences set A of T. A is then self-joined using the $1NN$-join function which itself uses the distance profile D to obtain the similarity join set J_{AA} which contains pairs of all nearest neighbours of all sub-sequences $T_{i,m}$ in T. Note that since we are joining A with itself, an exclusion radius $r > 1$ must be provided to avoid finding the trivial solution that $T_{i,m}$ is actually the nearest neighbour of $T_{i,m}$. Having obtained J_{AA}, a matrix profile P_{AA}, which is the vector containing all distances of all pairs in JA_{AA} can now be calculated using J_{AA} and D. Using the self similarity join matrix profile index I_{AA} allows fast access of the nearest neighbour of $A[i]$ by accessing $I_{AA}[i]$.

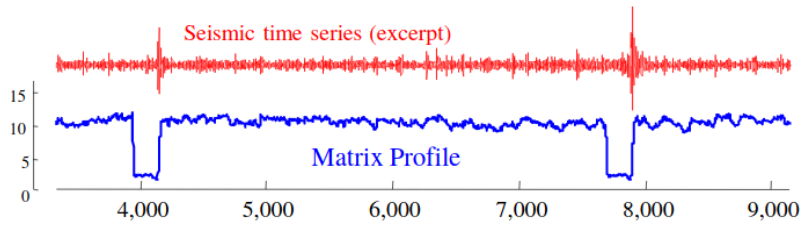


Figure 4: Example of a matrix profile [63]

Several algorithmic solutions exist for efficient motif discovery via matrix profiles, among others:

- MASS for exact solutions [63]
- STAMP for approximate solutions [63]
- STOMP for exact solutions [65]
- SCRIMP++ for exact or approximate [64]

For this work, we decided to use SCRIMP++ for its excellent run time, flexibility and because of its availability as Python package [31]. Hence for brevity, only SCRIMP++ will be described in detail below.

SCRIMP++ The SCRIMP++ algorithm is two-stage $O(n^2)$ algorithm (Fig. 5) that is capable of detecting fixed length motifs in either approximate or exact fashion. The first stage of SCRIMP++ is an ultra-fast preprocessing algorithm called PreSCRIMP (fig. 55). PreSCRIMP uses a property referred to as the Consecutive Neighborhood Preserving (CNP) Property of time series sub-sequences, which states that a matrix profile index (I_{AA}, see Table 6) can be split into sections such that *"within each section, a set of consecutive sub-sequences find another set of consecutive sub-sequences as their nearest neighbors"* [64]. This means that for sub-sequences $T_{i,m}$ and $T_{i+1,m}$, their nearest neighbours are $T_{j,m}$ and $T_{j+1,m}$ with a very high probability. Exploiting the CNP property, PreSCRIMP produces a very close approximate matrix profile very fast. It samples sub-sequences from time series at a fixed interval and then for each of the sampled sub-sequences, its exact nearest neighbour is found. If $T_{i,m}$ is a sampled sub-sequence with nearest neighbour $T_{j,m}$, then according to the CNP property, the nearest neighbour of $T_{i+k,m}$ is $T_{j+k,m}$, $(k = -s+1, -s+2, ..., -2, -1, 1, 2, ..., s-2, s-1)$. PreSCRIMP computes the distances between these pairs of sub-sequences and updates the matrix profile if a smaller distance value shows up. Based on the matrix profile produced by PreSCRIMP, the SCRIMP algorithm then updates the matrix profile until either a run time threshold t or an exact solution is reached. SCRIMP does that by first computing a z-normalized euclidean distance matrix for each pair

of time series sub-sequences $T_{i,m}$ and $T_{j,m}$:

$$d_{i,j} = \sqrt{2m(1 - \frac{Q_{i,j} - m\mu_i\mu_j}{m\sigma_i\sigma_j})}$$

Based on the distance matrix, the algorithm evaluates the distance matrix row-by-row in-order and updates the matrix profile accordingly.[31] See section A.1 for pseudo code.

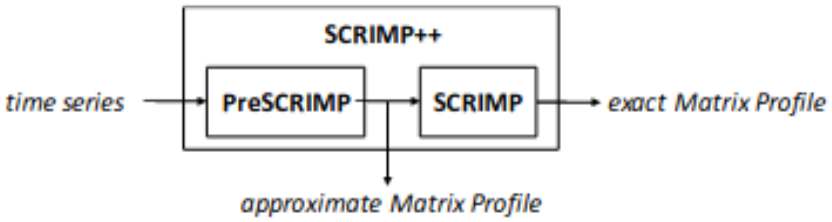


Figure 5: Illustration of SCRIMP++ [64]

Unlike the originally proposed algorithm from 2002 [35], SCRIMP allows the exact detection of motifs and its low run time complexity allows to scan for a large number of motif lengths. It still suffers from being unable to detect scaled motifs, though.

In this work, we used matrixprofile-ts, an existing Python implementation of the SCRIMP++ algorithm [31].

3.3.2 Grammar Induced Motif Discovery

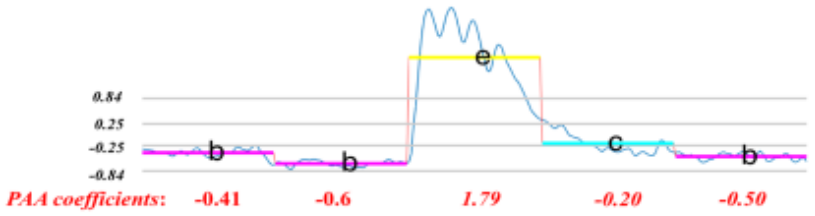
Most motif discovery algorithms consider the length of a motif as important hyperparameter that must be inferred from domain knowledge. In order to overcome this shortcoming, Lin et al. [19] introduced a new approach in 2018 that first discretizes a time series into its Symbolic Approximate Aggregation (SAX) representation [32] and then uses grammar rules to infer variable-length motifs.

SAX Obtaining the SAX representation of a z-normalized time series C of length n is a two step process. In a first step, PAA is used to summarize sub-sequences of length w according to the following formula: [32]

$$\bar{c}_i = \frac{w}{n} \sum_{j=\frac{n}{w}\cdot(i-1)+1}^{\frac{n}{w}\cdot i}$$

The resulting time series $\bar{C}$ is then discretized in the second step by mapping PAA to a equally sized areas under a Gauss curve (z-normalized time series are

normally distributed) and assigning each are a symbolic string representation from an alphabet of length a. These areas $B = \beta_1, ..., \beta_{a-1}$ are chosen such that the area under a N(0,1) Gaussian curve from β_i to $\beta_{i+1} = \frac{1}{a}$ (β_0 and β_a (Fig. 6) represent positive and negative infinity).[32]



β_i \ a	2	3	4	5
β_0	0	-0.43	-0.67	**-0.84**
β_1	-	0.43	0	**-0.25**
β_2	-	-	0.67	**0.25**
β_3	-	-	-	**0.84**

Figure 6: Example of Generating a SAX word [19]

In this work, we will use the SAX generation algorithm proposed by Lin et al. [19], FastSAX, because of its efficient computation and because it is available in the already existing JAVA implementation of HIME upon which our implementation is based [57]. FastSAX follows a dynamic programming approach where a sub-sequence is skipped if its SAX representation is identical to the last recorded one, saving computation time. See A.3 for pseudo code.

Induction Graph In order to reduce the number of SAX words that need to be tested in the discovery of long motifs (numerosity reduction (NR)), Lin et al. introduced a so called induction graph.

"[The] Induction Graph (Fig. 7), a graph structure that helps determine the order of scanning and enumeration of motif candidates during motif discovery. Each node contains 2 edges (next edge and forward edge). Two nodes connected by these two edges are denoted as S_i^{next} and $S_i^{forward}$ (Fig. 7), these are the black and yellow arrows, respectively). S_i next is the node representing the sub-sequence $s_{i+1,i+l}$ and $S_i^{forward}$ is the next non-similar sub-sequence determined by numerosity reduction [...]. The Induction Graph can be stored by only recording the nodes connected by $S_i^{forward}$."[19]

This definition of the Induction Graph offers several desirable properties: as already mentioned, it allows for an efficient numeriosity reduction during the motif discovery process by only testing nodes that are connected via $S_i^{forward}$. Furthermore, it also allows for an efficient storage, since only the nodes connected

via $S_i^{forward}$ need to be stored, all other edges and nodes can be reconstructed using these nodes.

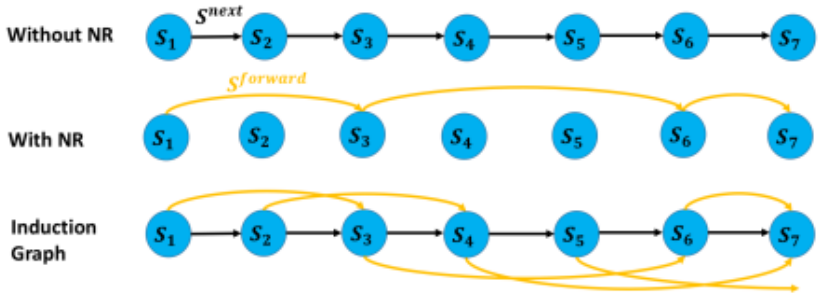


Figure 7: Illustration of Induction Graph [19]

Hierarchical based Motif Enumeration (HIME) The HIME algorithms mechanism is best explained by the summary of Lin et al:

"Intuitively, the algorithm conducts a left to right passing through all nodes via the next edge in Induction Graph G. For each node, the algorithm recursively executes two major functions: RecursiveEnumeration [...] and RemoveCovered-Motif [...]. In RecursiveEnumeration step, SAX words are formed to represent variable-length sub-sequences and to detect repeating sub-sequences. Remove-CoveredMotif removes short motifs found that are completely covered by longer motifs to maintain a small size of motif set at a low cost. The motifs detected by the algorithm are stored in Motif Set [...]. Finally, post-processing is applied to remove trivial and false-positive motifs candidates [...]. Note that HIME can also utilize numerosity reduction by only examining the nodes [connected via $S_i^{forward}$*] G."*[19]

The description of the HIME algorithm shows where its efficiency and versatility come from: because short motifs that are covered by longer motifs are discarded from the motif set, only the most relevant motifs are detected, independent of their length, and if numeriosity reduction is applied by only examining nodes connected via $S_i^{forward}$, the process of motif discovery can be sped up further, albeit at the cost of risking that some patterns might be missed. See A.2 for pseudo code.

In this work, we used the Python-wrapper motif-classify [37] for the SAX/HIME-based motif discovery JAVA library [57].

3.4 Classification Algorithms

Which classification algorithm is suited best for a given problem setting is a non-trivial question on its own. Hence for this work, we chose not to do an in-depth analysis and reason about the relevant parameters, but instead opted to take recourse to research conducted by other researches. Allouch et al. whose work on RoadSense bears striking similarities to this project but uses a different dataset, tested a C4.5 decision tree, a SVM and a Naive Bayes classifier, of which the C4.5 decision tree performed best [2]. Janko et. al, as part of the Sussex-Huawei Locomotion challenge, tested k-nn, XGB, Random Forest, SVM algorithms as classifiers and found that XGB performed best and k-nn performed worst [29]. Based on these findings, we chose to test Random Forests, CART-Trees and SVM classifiers and experimentally find which classifier works best for the given feature extraction methods.

Because the focus of this work is on comparing feature extraction and selection algorithms, only a short overview of the used algorithms is provided.

3.4.1 Random Forest (RF)

Random forests are built by using combinations of tree predictors created from a vector of random values sampled independently from the same distribution for all trees in the forest during the training [7] . During the inference, as illustrated by Fig. 8, the individual tree predictors classify a random sample of features and the forest then decides by majority vote on the final class. Regarding the generalization error, Breiman states the following:

The generalization error for forests converges a.s. to a limit as the number of trees in the forest becomes large. The generalization error of a forest of tree classifiers depends on the strength of the individual trees in the forest and the correlation between them. Using a random selection of features to split each node yields error rates that compare favorably to [existing algorithms], but are more robust with respect to noise. Internal estimates monitor error, strength, and correlation and these are used to show the response to increasing the number of features used in the splitting."[7]

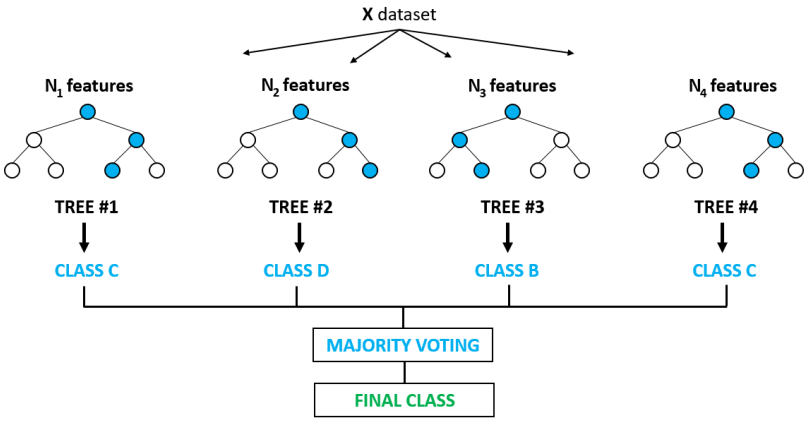


Figure 8: Symbolic Random Forest[54]

In this work, the Sklearn implementation of the Random Forest classifier was used [45].

3.4.2 CART-Tree

The CART tree is a decision tree classifier composed of pruned nested binary sub-trees. The training and construction phase of the CART tree is best described by Steinberg:

"The CART decision tree is a binary recursive partitioning procedure capable of processing continuous and nominal attributes as targets and predictors. Data are handled in their raw form; no binning is required or recommended. Beginning in the root node, the data are split into two children, and each of the children is in turn split into grandchildren. Trees are grown to a maximal size without the use of a stopping rule;essentially the tree-growing process stops when no further splits are possible due to lack of data. The maximal-sized tree is then pruned back to the root (essentially split by split) via the novel method of cost-complexity pruning. The next split to be pruned is the one contributing least to the overall performance of the tree on training data (and more than one split may be removed at a time). The CART mechanism is intended to produce not one tree, but a sequence of nested pruned trees, each of which is a candidate to be the optimal tree. The "right sized" or "honest" tree is identified by evaluating the predictive performance of every tree in the pruning sequence on independent test data"[59]

The CART tree is distinct from the Random Forest however in that after training, no forest of independent trees exists where each of the trees votes for a class during inference and the final class is decided by majority vote, but instead the

composition of nested sub-trees acts as a whole like a classical decision tree, see Fig. 9.

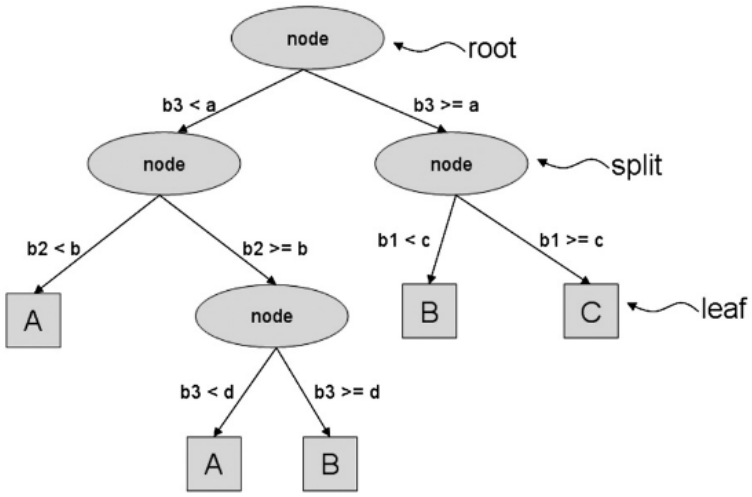


Figure 9: Symbolic Decision Tree [8]

In this work, the Sklearn implementation of the CART classifier was used [46]

3.4.3 Support Vector Machine (SVM)

The support vector machine is a classical machine learning approach. Assuming that input vectors are given that are not linearly separable, the machine conceptually implements the following mechanism: the input vectors are mapped to a high dimensional feature space using a non-linear function such that in this newly constructed feature space, a linear decision surface can be found [14], see Fig. 10. Regarding the generalizability of such created models, Cortes states that the *"[...] special properties of the decision surface ensures high generalization ability of the learning machine."* [14]

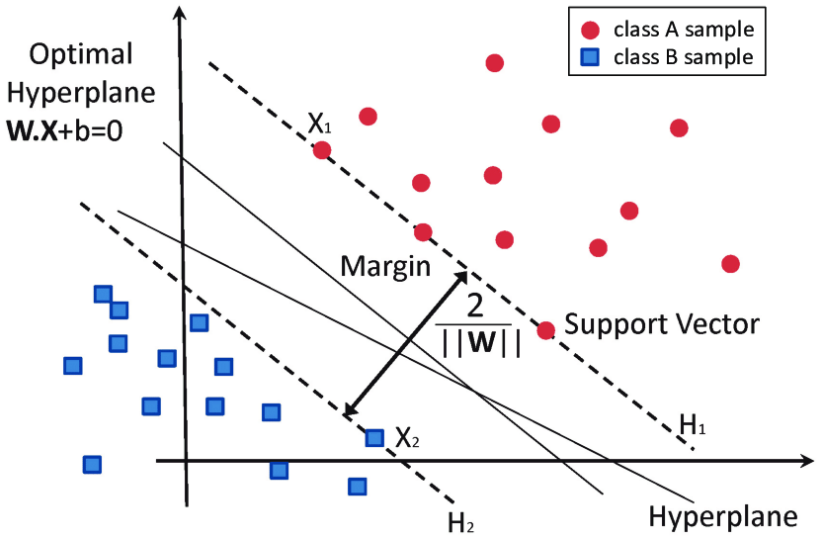


Figure 10: Symbolic SVM[20]

In this work, the Sklearn implementation of the SVM classifier was used [47]

3.4.4 Multi Layered Perceptron (MLP)

The multi layered perceptron is a model built in a similar fashion as neurons in the brain work: each perceptron, resembling a an abstracted neuron, has an input, and output and an activation function. A layer of a multilayered perceptron consists of multiple individual perceptrons and multiple layers are stacked upon another a multi layered perceptron. In the most basic case of a feed-forward network, each nodes output in each layer is connected with each nodes input in each subsequent layer, as illustrated by Fig. 11. Regarding the required number of such created layers, Marius stated that:

"Introduction of several layers was determined by the need to increase the complexity of decision regions. As shown in the previous paragraph, a perceptron with a single layer and one input generates decision regions under the form of semi planes. By adding another layer, each neuron acts as a standard perceptron for the outputs of the neurons in the anterior layer, thus the output of the network can estimate convex decision regions, resulting from the intersection of the semi planes generated by the neurons. In turn, a three-layer perceptron can generate arbitrary decision areas."[38]

This means that already a relatively simple three-layer perceptron is a good start for any classification problem, although more complex architectures or special feature engineering techniques could be required for more complex tasks

such as the classification of data points for which the IID assumption does not hold.

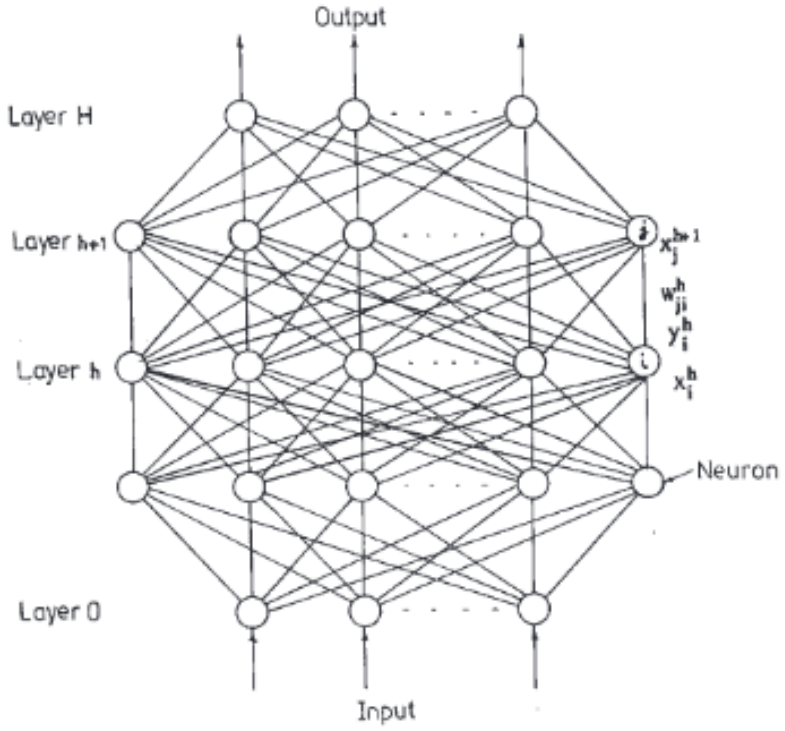


Figure 11: Symbolic MLP [44]

In this work, the Sklearn implementation of the MLP classifier was used [48]

4 Methodology

After an extensive literature survey that yielded the above mentioned approaches, algorithms and libraries, we decided to implement the project in Python for the languages general flexibility, availability of pre-existing machine-learning packages as well as its general prominence in the data science world.

First we roughly designed a software architecture as abstract as possible and then a top-down development approach was chosen, combined with rapid software prototyping in Jupyter Notebook [15]. The prototyped and tested functions where then migrated to the general software architecture.

Finding the best overall classifier is then achieved by hyperparameter optimization and comparing the performance of the classifiers with one another. hyperparameters are parameters that are passed to a model during creation and they affect the way the model is created. Because it can only be estimated beforehand which set of parameters creates a model ideal for the given task and dataset, the need for optimization arises, which we achieved as follows. For the given sets of classifiers $C = \{RF, CART, SVM\}$, feature extraction and selection methods $F = \{FRESH, SCRIMP++, SAX/HIME\}$ and hyperparameter tuples $H = \{(c_i, f_j), i, j \in \mathbb{N}\}$ where c_i represents a set of hyperparameters for a classifier and f_i represents a set of hyperparameters for a feature extraction method, we evaluated their cross product $C \times F \times H$ experimentally and chose the combination that yielded the lowest validation error. To reduce computation time, tuples $(c_i, f_j) \in H$ were sampled from specified distributions (Random Search) instead of trying out all possible hyperparameters (Grid Search).

4.1 Implementation

4.1.1 Software Engineering

The implementation adheres to the best of the authors ability to the internal and external quality characteristics described in [40], with a particular focus on the external characteristics correctness, efficiency and adaptability and the internal characteristics flexibility, readability and testability because a focus on these characteristics will be most advantageous when adapting the pipeline for different data sets, tasks and optimizations such as parallelizations in future work while at the same time guaranteeing reliable and actionable results. This is achieved by the use of defensive programming [40], adherence to classic OOP principles [40] as well as the application of design and architecture patterns [39] and general best practices (documentation, style guides according to PEP8 [26]) where seen fit.

Architecture The general architecture of the implementation follows a pipeline pattern (Fig. 12). A pipeline is an aggregation of a data access object (DAO Pattern [42]), a preprocessor, a feature extractor and a model factory (Factory and Strategy Pattern[18]). These components execute their operations and pass the results down the pipeline via a well-defined interface. The description of the components as well as the UML diagram (Fig. 60 in Appendix C.1) is kept to an abstract level because the concrete implementations of these classes may vary dependent on the dataset and task at hand.

DAO The DAO has the responsibility to handle all disk IO: reading the data set from the disk, storing and loading extracted features and storing and loading trained models.

Preprocessor The preprocessor receives the data set from the DAO handles all preprocessing steps that are defined by the user: normalization, segmentation and de-segmentation, outlier removal, time-unit conversion, re-sampling, labeling and projecting accelerometer data from the coordinate system of the mobile device either to global vehicle coordinates or to a dimensionality reduced space.

Feature Extractor The feature extractor receives the preprocessed data from the preprocessor and extracts and selects the features generated by any supervised feature extraction and selection algorithm. It's important to note that the feature extractor is only capable to selection by filtering [43]. Should a wrapper [55] be required, the extractor has to be used in conjunction with the model factory. The extracted features are handed over to the DAO by the pipeline for persistent on-disk storage.

Model Factory The model factory receives the extracted features from the feature extractor and either creates a specified model, optimized for a given hyper-parameter search space, or finds the optimal model of a range of given different models. The optimized classifier is handed over to the DAO by the pipeline for persistent on-disk storage.

Pipeline Facade The interaction between the components of the pipeline is commanded by the pipeline facade.

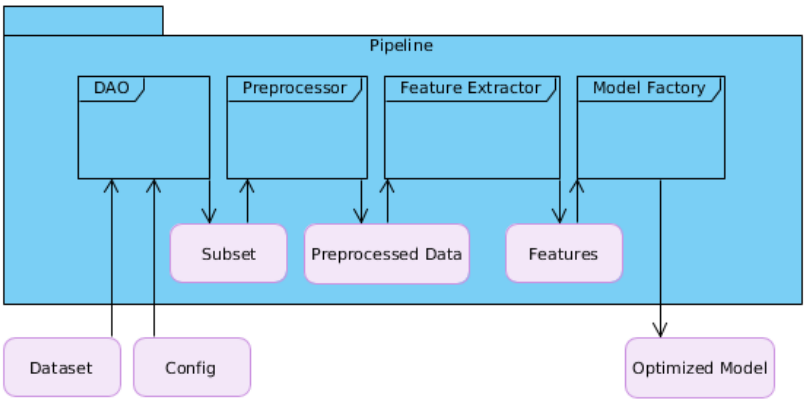


Figure 12: Architecture Overview

4.1.2 Preprocessing

The preprocessing is done in an implementation of the abstract preprocessor that is tailored to the specific requirements of the Sussex-Huawei dataset and

closely follows the steps taken by [2, 53, 58]. The preprocessing executes the following steps in order:

1. Convert all time units from UNIX time to Datetime format.

2. Combine the target vectors (labels) y_{rt} for road type and y_{at} for activity type and the feature matrix (xyz-accelerometer data) X to a single matrix by appending y_{rt} and y_{at} as columns to X (if in training mode, for inference, this step is skipped), i.e. $X_c = X \bowtie_{index} y_{rt} \bowtie_{index} y_{at}$

3. Remove corrupted rows (NaN, Null, etc) from X_c.

4. Split X_c in training and test set (if in training mode, for inference, this step is skipped). The following steps will be executed for both training and test sets and both sets will be referred as X_c.

5. Segment the X_c row-wise by the label columns and remove segments that do not have the target combination of labels $y_{at} = Car$ and $y_{rt} \in \{Country, City\}$. (if in training mode, for inference, this step is skipped).

6. Resample each segment $s_c \subset X_c$. The Sussex-Huawei datasets accelerometer data is sampled at 100Hz, which means that it contains many potentially irrelevant high frequency noise components. Furthermore, this will also reduce the cardinality of the dataset. The resampling is implemented by representing sub-sequences of the time-series segments with their mean, i.e
$$s_c^r(i) = \frac{1}{f} \sum_{j=i \cdot f}^{f \cdot (i+1)} s_c(j)$$
with $i \in [0, \frac{n}{f}]$ for frequency f and size n of s_c. The resampled segments are then stored in X_c^r

7. Dimensionality reduction of each segment is achieved by projecting the 3-dimensional space of the accelerometer measurements on a 1-dimensional space. This done either using the euclidean norm or via PCA. For euclidean, this equals to
$$s_c^r(i)(k) = \sqrt{\sum_{j \in \{x,y,z\}} s_c^r(i)(j)^2}$$
where k denotes the newly introduced dimension. For PCA this equals to
$$s_c^r(i)(k) = \Sigma_{s_c^r} \times \Lambda_{s_c^r} \times \Sigma_{s_c^r}^T (max(\lambda_e))(i)$$
where k denotes the newly introduced dimension, $\Sigma_{s_c^r}$ denotes the covariance matrix of s_c^r, $\Lambda_{s_c^r}$ denotes the matrix of the eigenvalues of $\Sigma_{s_c^r}$ and $max(\lambda_e$ denotes our choice of the largest eigenvalue, i.e. the column of $\Sigma_{s_c^r} \times \Lambda_{s_c^r} \times \Sigma_{s_c^r}^T$ that explains the variance of the dataset best. The old xyz-dimensions are removed after each reduction operation.

8. Statistical outlier removal by removing the p and $(1-p)$ quantiles of each time-series segment.

9. Desegementation by row-wise concatenation of the segments s_c^r, so all segments are stored again in a single matrix X_c^r (inverse operation to step 5).

10. Split X_c^r column-wise into the new feature matrix X_n and label vector y_n (inverse operation to step 2). X_n and y_n can then be handed downwards in the pipeline for further feature engineering or, at least theoretically, directly handed to a classifier.

4.1.3 Feature Engineering

For the training phase, two different backward feature selection technique are applied after the three different feature extraction approaches:

- Filtering [43] (Fig. 14). is a technique where out of N different features, M features are selected based on some kind of quality metric. FRESH used statistical independence and FDR tests for filtering after extracting all possible features (see 3.2).

- The wrapper technique [55] (Fig. 13), which was used for the SAX/HIME and SCRIMP++ feature extraction approaches, is applied in case no quality metric for the extracted features can be defined and only by training a classifier using the extracted features and evaluating the classifiers performance, their suitability for the task can be determined. This means N features f_n are extracted and then for each feature $f_n \in N$, a classifier C must be trained an evaluated.

The type of features that were found relevant are stored on disk and are used during inference to specifically extract these features.

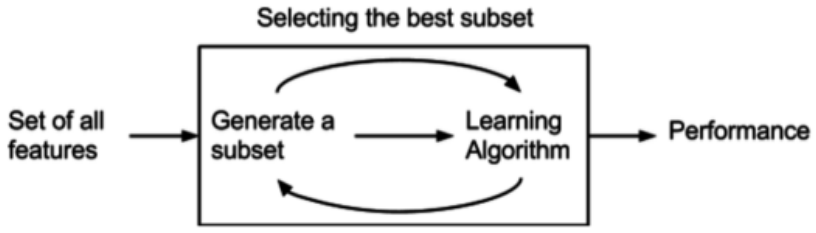


Figure 13: Feature selection via wrapping [13]

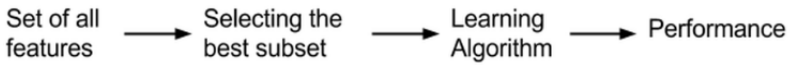


Figure 14: Feature selection via filtering [13]

"Manual" Statistical Feature Extraction and Selection Manually extracted and selected statistical features are extracted by directly coding the necessary operations in Python. For a given PAA length l, the sub-sequences are aggregated using summary statistics. For each sub-sequence s_i^l, a row of the following shape is added to the feature matrix:

$min(s_i^l)$	$max(s_i^l)$	$mean(s_i^l)$	$std(s_i^l)$	$var(s_i^l)$

TSFRESH The TSFRESH Python package [10] which implements the FRESH algorithm discussed in section 3.2 offers two API methods, one for feature extraction and one for the selection of the relevant extracted features, which are applied on the preprocessed dataset X_c^r. In the extraction step of the training phase, all possible features are extracted by invoking the extraction method without any parameters specifying which feature should be extracted and in the selection step, the relevant features are selected for the given FDR (backwards-selecting filter method [43], Fig. 14)). The names of the relevant features are then stored on disk. During the inference phase, the names of the features that were selected as relevant features during a previous training phase are loaded from disk by the pipeline and then handed to TSFRESH extraction method so that only these features are extracted.

TSFRESH extracts n statistical features for m different time series, resulting in a feature matrix F of cardinality m and dimensionality n. Optionally, in order to ensure that even for trips of variable length the statistical features are comparable, each trip can be segmented in to equally long segments of length l before it is handed to TSFRESH and segments with inhomogeneous labeling can be discarded. See 5.1.3 for details.

The TSFRESH API calls are wrapped in a feature extractor class as described in section 4.1.1 in order to ensure that the return values match the well-defined interface required by the pipeline object.

Matrix Profiles Motifs as features are engineered via matrixprofile-ts implementation of the SCRIMP++ algorithm [64, 31], for details see section 3.3.1. For each class $c \in C$, the indices $i_{m_c} \in I$ of the top motif m_c occurring in the preprocessed dataset X_c^r as well as the motifs distance d_{m_c} to its nearest neighbour is extracted via SCRIMP++ . Then each of its occurrences in homogeneously labelled sub-sequences of the time series is looked-up and the

sub-sequence s_{m_c} is summarized using its mean, standard deviation, minimum, maximum and distance to the nearest neighbour. Additionally, a tag is added for each motif. This results in a feature matrix with rows of the following shape, which is then used to build a classifier:

tag_{m_c}	$min(s_{m_c})$	$max(s_{m_c})$	$mean(s_{m_c})$	$std(s_{m_c})$	d_{m_c}

Furthermore, because we do not know in advance which motif length will be optimal for classification, motifs are extracted for various motif lengths and then their quality is assessed later while building the classifier (backwards-selecting wrapper method [55], Fig. 13)). Fortunately, parallelizing the extraction of motifs of different lengths is an embarrassingly parallel problem and could easily achieved with Pythons process based parallelism. During the inference phase, the same steps are applied except for the selection of sub-sequences s_c^r that do not have homogeneous labelling because this information is obviously not available during inference.

The motif extraction API calls are wrapped in a feature extractor class as described in section 4.1.1 in order to ensure that the return values match the well-defined interface required by the pipeline object.

SAX/HIME Motifs as features engineered via the motif-classify Python wrapper [37] for the SAX/HIME-based [19] motif discovery JAVA library GrammarViz [57] are extracted via a motif finder method provided by motif-classify that, fully parallelized, extracts motifs for various sliding window, PAA and alphabet sizes. The found motifs can then be handed over to a classifier. Because it is not clear beforehand which motifs are suited best for classification, a wrapper method similar to the one described for the features engineered via matrixprofile-ts has to be deployed.

Because the SAX/HIME approach did not yield actionable results during prototyping in Jupyter Notebook, it was not integrated in the software project. See section 6 for details on the results.

Anomaly Detection using Isolation Forests Before handing the engineered features over to the classification algorithm, the Isolation Forest anomaly detection algorithm [34] is used in a post-processing step for another round of outlier detection and removal. The detected abnormal rows are removed from the feature matrix produced by any of the feature engineering algorithms.

The Isolation Forest algorithm is an algorithm that works on two stages: during the training stage, Isolation Trees are built using sub samples of a training set, and then, during the testing stage, the test instances are passed through the Isolation Trees to obtain an anomaly score fore each instance. An Isolation Tree is a proper binary tree that recursively divides a given input dataset X by

randomly selecting an attribute q and split value p until either the tree reaches a height limit or all data points in the remaining data set X_{split} have the same values [34].

The Sklearn implementation of the Isolation Forest algorithm is used in this work [49].

4.1.4 Classification

For each of the classifiers $c_k \in C$ mentioned in section 3.4, a random search algorithm is applied to determine the optimized hyperparameters. The random search algorithm receives a distribution $D_{k,i}$ for each hyperparameter $h_{k,i}$ and then executes the following steps for N iterations until the optimal classifier is found:

1. For $h_{k,i}$ $D_{k,i}$ draw a value from the distribution
2. Train classifier c_k^n, $n \in N$ on the derived hyperparameter set $\{h_{k,1}, ..., h_{k,m}\}$
3. If $score(c_k^n) > score(c_k^{optimal})$ then $c_k^{optimal} = c_k^n$

After finding the optimal set of hyperparameters for each classifier, the classifiers $c_k^{optimal} \in C^{optimal}$ are compared with one another to determine which classifier is best. This classifier c_k^{best} is then stored on disk for later usage during inference tasks.

In this work, the Sklearn implementation of the random search algorithm was used [50].

5 Experiments

5.1 Setups and Constraints

The experiments were conducted on a machine with 96GB RAM with and an Intel(R) Xeon(R) Gold 6130 CPU with 16 physical cores operating at 2.1GHz and 32 threads (details see C.2.5). Given these constrains, the experiments were run using a 25% subset of the full dataset that was chosen using a roughly equal distribution of trip samples. In order too ensure that the experiments results are comparable, equal parameters were used for the preprocessing steps during each experiment and the 25% subset used remained the same. The chosen score for comparing the classifiers was the accuracy score as implemented by sklearn [51], which is given by

$$accuracy(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} 1(\hat{y}_i, y_i)$$

where y and $\hat{y}$ are the true and the predicted label vectors, y_i and $\hat{y}_i$ are the true and predicted labels for the i-th sample and $1(a, b)$ is the indicator function

defined as $1(a, b) = 1$ if $a = b$, 0 otherwise. The same score was used throughout all experiments.

5.1.1 Preprocessing Parameters

The preprocessing steps described in section 4.1.2 were executed before all experiments using the following variables:
TR... Training set size
TE... Test set size
TV... Validation set size
OQ... Outlier removal quantile
RS... Resampling frequency
DR... Dimensionality reduction

TR	TE	TV	OQ	RS	DR
0.5	0.25	0.25	0.1	10Hz, 1Hz	PCA, Euclidean

This yielded four different combinations of preprocessing parameters which resulted in four different experimental setups (Setup 10 Hz Euclidean Norm to 4) that had to be evaluated individually for each feature extraction and classifier combination (see section 5.2).

- Setup 10 Hz Euclidean Norm: Resampling to 10Hz, Dimensionality reduction via Euclidean Norm
- Setup 1 Hz Euclidean Norm: Resampling to 1Hz, Dimensionality reduction via Euclidean Norm
- Setup 10 Hz PCA: Resampling to 10Hz, Dimensionality reduction via PCA
- Setup 1Hz PCA Resampling to 1Hz, Dimensionality reduction via PCA

In section Results (6), the results for each of the four experimental setups will be discussed an the results will be compared in Conclusion section (7).

The low resampling frequency had to be chosen for the excessive memory consumption produced when using larger values. The chosen resampling rate will allow the detection of signals with a frequency of 5 Hz or smaller [56].

5.1.2 Train, Test and Validation Split

All experiments were conducted using a training, test and validation set with the following sample distributions:

Set	City	Country Road
Train	53.8%	46.1%
Test	52.1%	47.9%
Validation	54.0%	46.0%

Train, Test and Validation set were preprocessed independently.

5.1.3 Simulated Change Point Detection (SCPD)

One problem when applying PAA to time series in classification problems is that labels might not be homogeneously aligned with the segments that are aggregated, which can negatively affect classification quality. One way to tackle this problem is to detect so called change points [4], points in the time series where it switches from one Label to another, and use this information to ensure that the data points that are aggregated are of the same class.

Because Change Point Detection (CPD) is not within scope of this work, Simulated Change Point Detection (SCPD) is introduced for experimental purposes. SCPD simply uses the labeling information available for the Training, Test and Validation sets to ensure that all segments aggregated by PAA are homogeneously labeled by ignoring or removing heterogeneously labeled sub-sequences. As shown in the experiments, SCPD significantly affects the classification results.

5.2 Hyperparameter Search Spaces

For each combination of classifiers and feature extraction/selection methods (see section 4 for general methodology), the following hyperparameter search spaces are evaluated.

5.2.1 RF

K... K-fold cross validation
N... Number of estimators
I... Number of iterations (combinations drawn from distribution)
D... Max depth of tree
S... Min samples required for split
B... Bootstrapping
C... Split criterion

Type	K	I	N	D	S	B	C
Fixed Value	5	50					
Variable						t, f	gini, entropy
Range: From			1	1	2		
Range: To			128	128	20		

5.2.2 CART-Tree

K... K-fold cross validation
I... Number of iterations (combinations drawn from distribution)
D... Max depth of tree
S... Min samples required for split
SP... Splitter
C... Split criterion

Type	K	I	D	S	SP	C
Variable	5	50			best, random	gini, entropy
Range: From			1	2		
Range: To			128	20		

5.2.3 SVM

K... K-fold cross validation
I... Number of iterations (combinations drawn from distribution)
D... Degree
G... Gamma exponent (range from 1^{-G} to 10^{G-1})
C... C
M... Max iterations
K... Kernel
S... Shrinking
P... Probability

Type	K	I	G	D	K	M	S	P
Variable	5	50	10		rbf, poly		t, f	t, f
Range: From				2		2		
Range: To				30		5000		

5.2.4 MLP

K... K-fold cross validation
I... Number of iterations (combinations drawn from distribution)
S... Error back propagation solver
M... Maximum number of iterations of solver
AE... Alpha exponent (range from 1^{-A} to 10^{A-1})
A... Architectures (hidden layers)
AF... Activation functions
L... Learning rate
LE... Learning rate initial exponent SH... Shuffle E... Early stopping

Type	K	I	S	M	AE	A	AF
Variable	5	50	adam lbfgs sgd		10	(32,16) (16,8) (8,4)	logistic relu tanh
Range: From				1			
Range: To				250			

Type	L	LE	SH	E
Variable	constant invscaling adaptive	10	t,f	, tf
Range: From				
Range: To				

5.2.5 TSFRESH

Segment length	Features	FDR
15	All	0.01
30	All	0.01
60	All	0.01
90	All	0.01

5.2.6 SAX/HIME

All possible combinations of the values in the Table were examined.

FastSAX Windows Size	FastSAX PAA Size	FastSAX Alphabet Size
8	3	6
16	4	8
32	5	10
64	6	

5.2.7 Matrix Profiles

All possible combinations of the values the Table were examined.

Motif Lengths	Motif Radii
10	1
25	5
50	10
100	100
150	150
200	200

6 Results

The Figures in the results section show a summary of results in each setup in each experiment. Because of the large number of experiments and the schema of the figures is similar in experiment, the figures will be explained once here.

For the Manual Extraction and Selection without SCPD, Manual Extraction and Selection with Full SCPD, TSFRESH Extraction and Manual Selection with Full SCPD, TSFRESH Extraction and Selection with Full SCPD, the PAA = 15, 30, 60, 90 diagrams display the best accuracy of each classifier for a given PAA length in a bar plot, for example see Fig. 15. The PAA Summary diagram show a box plot of summarized classifier accuracy in dependence of PAA length and the Classifier Summary diagram shows a box plot of the results of each classifier for all PAA lengths. At the end of each experiment, an additional summary is provided, comparing the best classifiers found in each setup in a bar plot, see Fig. 19 for example.

For the SAX/HIME, the figures illustrate summary statistics for each setup (box plot, Fig. 34, a comparison best classifier found (Fig. 35, bar plot) and summary statistics of the classifier performance for the sanity check (Fig. 36, box plot).

For the experiment MP with Semi-SCPD, the dependency of the of the feature extraction method on two parameters (motif length and motif radius) made it necessary to illustrate each classifiers performance individually in each setup in dependence of the parameters, for example see Fig. 37. The first five diagrams in each figure show a bar plot that visualizes for a given motif radius the best accuracy achieved by the given classifier in dependence of the motif length. The Summary diagram shows a summary statistic in the form of a box plot for the classifiers accuracy in dependence of the motif radius. At the end of the MP with Semi-SCPD experiment, there is again a bar plot comparing the best classifiers found in each setup.

6.1 Manual Extraction and Selection without SCPD

6.1.1 Setup 10 Hz Euclidean Norm

For Setup 10 Hz Euclidean Norm, features were manually extract-selected for all PAA lengths except 90 (15, 30, 60, which equals 1.5, 3, and 6 seconds given Setup 10 Hz Euclidean Norms 10Hz sampling rate). For a PAA length 90, no homogeneously labeled sub-sequences could be found (empty diagram in Fig. 15). All classifiers performed similarly on the extracted features, yielding accuracy scores in the mid to upper 60% to upper 70% range. The best classifier found was a CART tree which achieved an accuracy score of 79.0% on the validation set for a PAA length of 30, which equals 3 seconds given Setup 10 Hz Euclidean Norms 10Hz sampling rate.

Classifier	Best PAA Length	Best Accuracy
MLP	60	72.2%
CART	30	79.0%
RF	30	73.9%
SVM	30	69.4 %

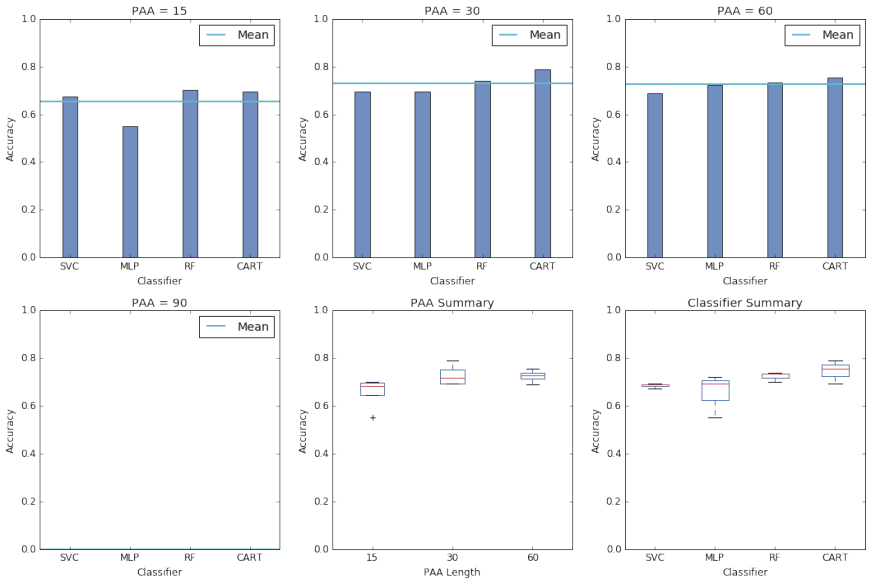


Figure 15: Results for Manual Extraction and Selection, 10Hz, euclidean norm

6.1.2 Setup 1 Hz Euclidean Norm

For Setup 1 Hz Euclidean Norm, features were manually extract-selected for all PAA lengths except 60 and 90 (15, 30, which equals 15 and 30 seconds given Setup 10 Hz Euclidean Norms 10Hz sampling rate). For PAA lengths of 60 and 90, no homogeneously labeled sub-sequences could be found. All classifiers performed similarly for their respective PAA length on the extracted features. For a PAA length of 15, they yielded accuracy scores in the upper 40% and lower 50% range and for a PAA length of 30, they yielded accuracy scores in the upper 50% and mid 60% range. The best classifier found was an SVM which achieved an accuracy score of 66.3% on the validation set for a PAA length of 30, which equals 30 seconds given Setup 1 Hz Euclidean Norms 1Hz sampling rate. Fig. 16 shows a summary of the results for this setup. The first row of diagrams displays the best accuracy of each classifier for a given PAA length. The PAA Summary diagram shows a box plot of summarized classifier accuracy

in dependence of PAA length and the Classifier Summary diagram shows a box plot of the results of each classifier.

Classifier	Best PAA Length	Best Accuracy
MLP	30	65.8%
CART	30	49.7%
RF	30	50.4%
SVM	30	66.3 %

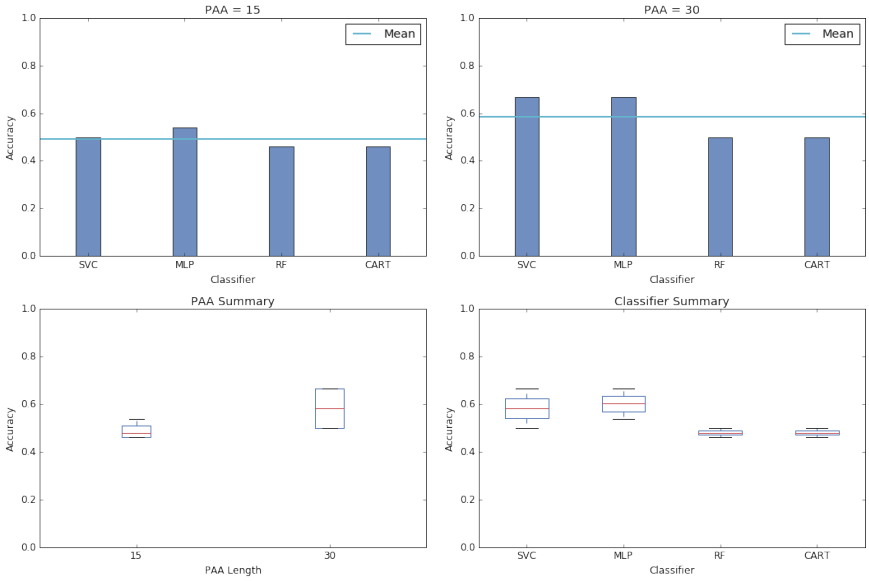


Figure 16: Results for Manual Extraction and Selection, 1Hz, euclidean norm

6.1.3 Setup 10 Hz PCA

For Setup 10 Hz PCA, features were manually extract-selected for all PAA lengths (15, 30, 60, 90 which equals 1.5, 3, 6 and 9 seconds given Setup 10 Hz PCAs 10Hz sampling rate). For PAA lengths of 60 and 90, no homogeneously labeled sub-sequences could be found. All classifiers performed similarly for their respective PAA length on the extracted features. For a PAA length of 15, they yielded accuracy scores in the upper 40% and lower 50% range and for a PAA length of 30, they yielded accuracy scores in the upper 50% and mid 60% range. The best classifier found was an SVM which achieved an accuracy score of 66.3% on the validation set for a PAA length of 30, which equals 3 seconds given Setup 10 Hz PCAs 10Hz sampling rate.

Classifier	Best PAA Length	Best Accuracy
MLP	90	68.3%
CART	90	71.7%
RF	15	64.6%
SVM	90	68.0 %

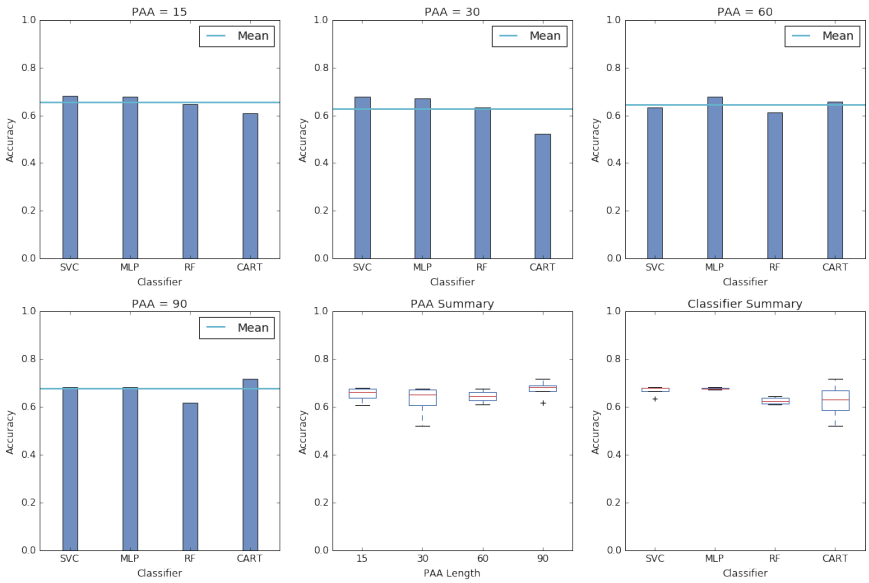


Figure 17: Results for Manual Extraction and Selection, 10Hz, PCA

6.1.4 Setup 1 Hz PCA

For Setup 1 Hz PCA, features were manually extract-selected for all PAA lengths except 60 and 90 (15, 30, which equals 15 and 30 seconds given Setup 10 Hz Euclidean Norms 10Hz sampling rate). For PAA lengths of 60 and 90, no homogeneously labeled sub-sequences could be found. All classifiers performed similarly for their respective PAA length on the extracted features. For a PAA length of 15, they yielded accuracy scores in the upper 40% and lower 50% range and for a PAA length of 30, they yielded accuracy scores in the upper 50% and mid 60% range. The best classifier found was an MLP which achieved an accuracy score of 66.1% on the validation set for a PAA length of 30, which equals 30 seconds given Setup 1 Hz PCAs 1Hz sampling rate.

Classifier	Best PAA Length	Best Accuracy
MLP	30	66.1%
CART	30	50.7%
RF	30	49.9%
SVM	30	66.0 %

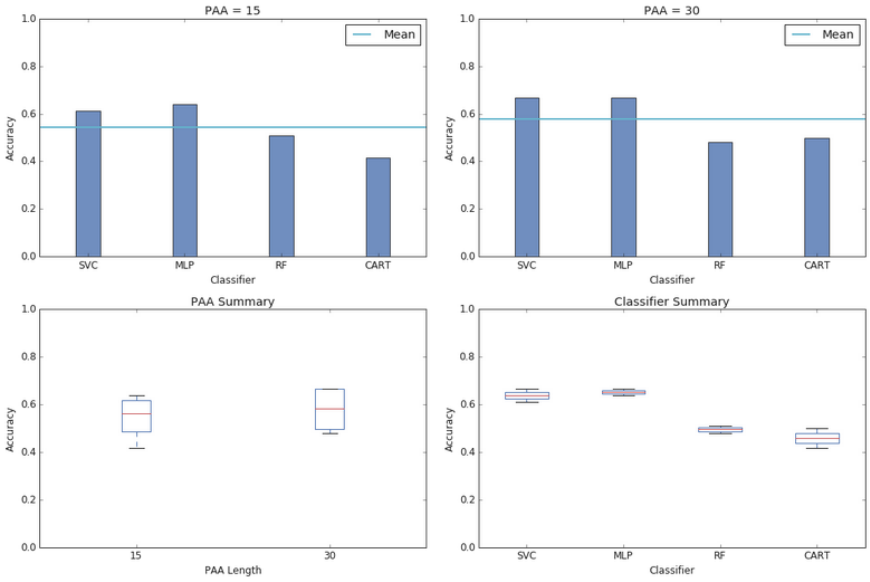


Figure 18: Results for Manual Extraction and Selection, 1Hz, PCA

6.1.5 Summary

The best classifier that could be built using manual extraction and selection without SCPD as feature engineering method showed to be a CART tree in combination with Setup 10 Hz Euclidean Norm (euclidean norm, 10Hz sampling rate, PAA length 30). Generally, the manual extraction-selection approach with manual turned out to be a quite good feature engineering approach, yielding an accuracy of 79.0% for the best classifier found.

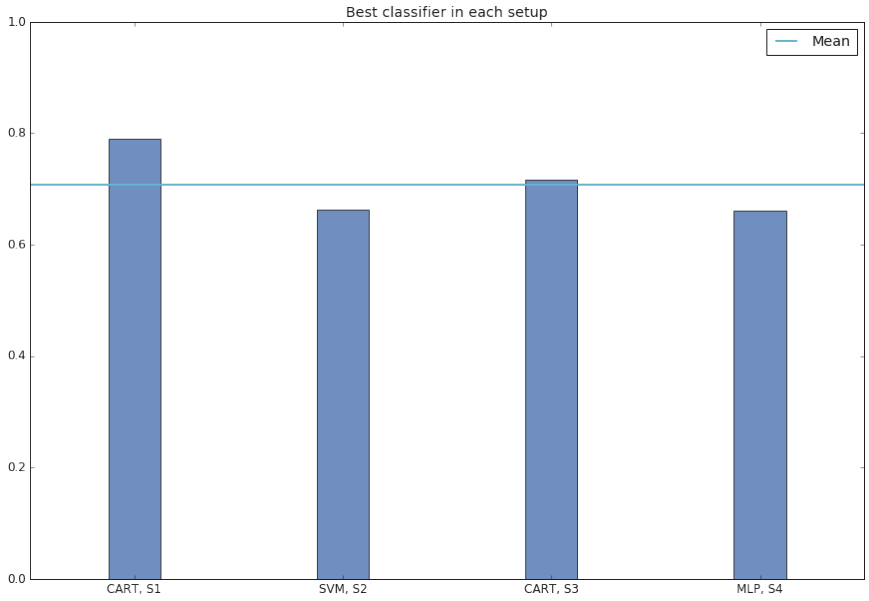


Figure 19: Summary of best classifiers in each setup

6.2 Manual Extraction and Selection with Full SCPD

6.2.1 Setup 10 Hz Euclidean Norm

For Setup 10 Hz Euclidean Norm, features were manually extract-selected for all PAA lengths except 90 (15, 30, 60, which equals 1.5, 3, and 6 seconds given Setup 10 Hz Euclidean Norms 10Hz sampling rate). For a PAA length 90, no homogeneously labeled sub-sequences could be found (empty diagram in Fig. 20). All classifiers performed similarly on the extracted features, yielding accuracy scores in the mid to upper 60% and mid to upper 70% range. The best classifier found was a CART tree which achieved an accuracy score of 78.9% on the validation set for a PAA length of 30, which equals 3 seconds given Setup 10 Hz Euclidean Norms 10Hz sampling rate.

Classifier	Best PAA Length	Best Accuracy
MLP	60	75.5%
CART	30	78.9%
RF	30	78.1%
SVM	60	77.2 %

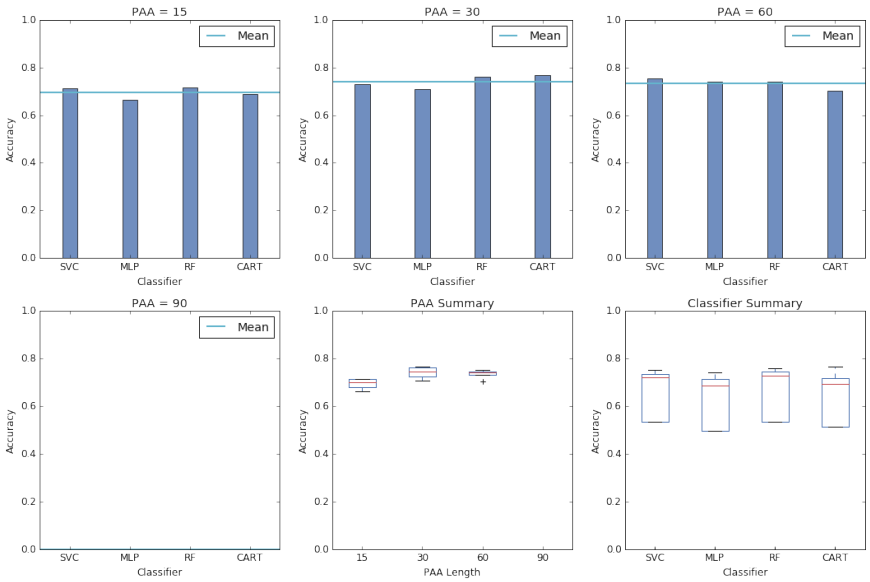


Figure 20: Results for Manual Extraction and Selection, 10Hz, euclidean norm

6.2.2 Setup 1 Hz Euclidean Norm

For Setup 1 Hz Euclidean Norm, features were manually extract-selected for all PAA lengths except 60 and 90 (15, 30, which equals 15 and 30 seconds given Setup 10 Hz Euclidean Norms 10Hz sampling rate). For PAA lengths of 60 and 90, no homogeneously labeled sub-sequences could be found. All classifiers performed similarly for their respective PAA length on the extracted features. For a PAA length of 15, they yielded accuracy scores in the upper 40% and lower 50% range and for a PAA length of 30, they yielded accuracy scores in the upper 50% and lower 70% range. The best classifier found was an MLP which achieved an accuracy score of 70.2% on the validation set for a PAA length of 30, which equals 30 seconds given Setup 1 Hz Euclidean Norms 1Hz sampling rate.

Classifier	Best PAA Length	Best Accuracy
MLP	30	70.2%
CART	30	60.8%
RF	30	69.9%
SVM	60	60.3 %

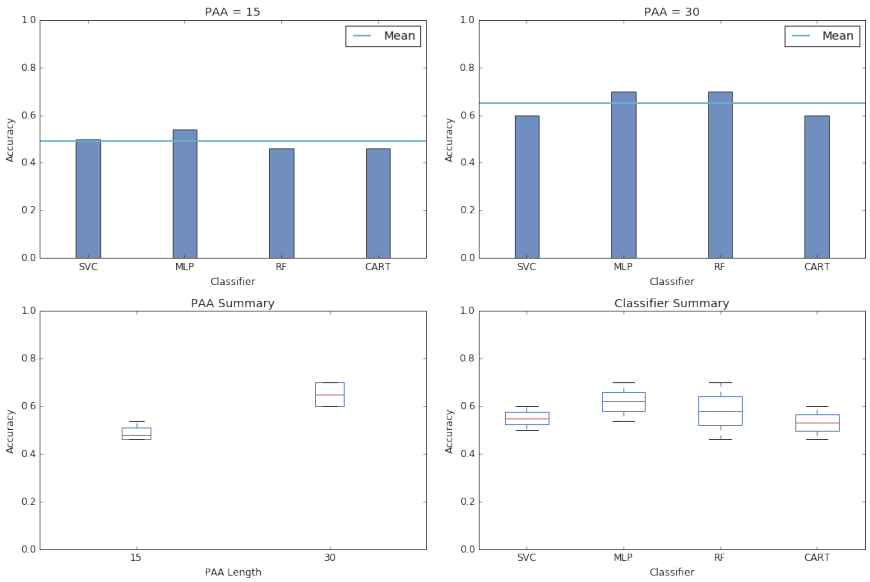


Figure 21: Results for Manual Extraction and Selection, 1Hz, euclidean norm

6.2.3 Setup 10 Hz PCA

For Setup 10 Hz PCA, features were manually extract-selected for all PAA lengths (15, 30, 60, 90 which equals 1.5, 3, 6 and 9 seconds given Setup 10 Hz PCAs 10Hz sampling rate). All classifiers performed similarly in their respective PAA length on the extracted features. Generally they yielded accuracy scores in the upper 50% and upper 70% range. Interestingly for a PAA length of 15, all classifiers yielded results very close to the mean of this PAA lengths accuracy scores. The best classifier found was an SVC which achieved an accuracy score of 70.2% on the validation set for a PAA length of 90, which equals 9 seconds given Setup 10 Hz PCAs 10Hz sampling rate.

Classifier	Best PAA Length	Best Accuracy
MLP	90	77.2%
CART	90	78.8%
RF	90	76.9%
SVM	90	80.3 %

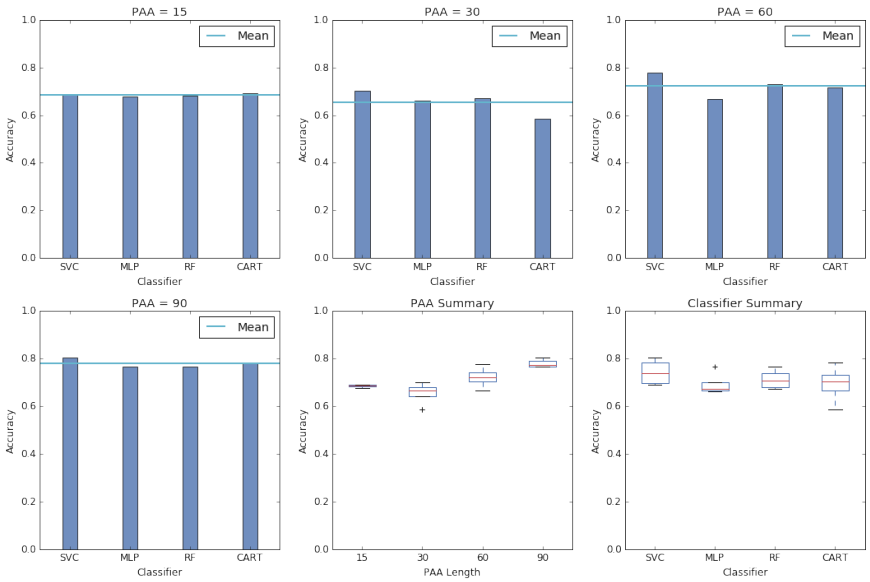


Figure 22: Results for Manual Extraction and Selection, 10Hz, PCA

6.2.4 Setup 1 Hz PCA

For Setup 1 Hz PCA, features were manually extract-selected for all PAA lengths except 60 and 90 (15, 30, which equals 15 and 30 seconds given Setup 10 Hz Euclidean Norms 10Hz sampling rate). For PAA lengths of 60 and 90, no homogeneously labeled sub-sequences could be found. For a PAA length of 15, the classifiers yielded accuracy scores in the lower 60% and mid 80% range and for a PAA length of 30, they yielded accuracy scores in the lower 60% and lower 70% range. The best classifier found was an MLP which achieved an accuracy score of 85.2% on the validation set for a PAA length of 15, which equals 15 seconds given Setup 1 Hz PCAs 1Hz sampling rate.

Classifier	Best PAA Length	Best Accuracy
MLP	15	85.2%
CART	15	77.8%
RF	30	70.1%
SVM	15	81.5 %

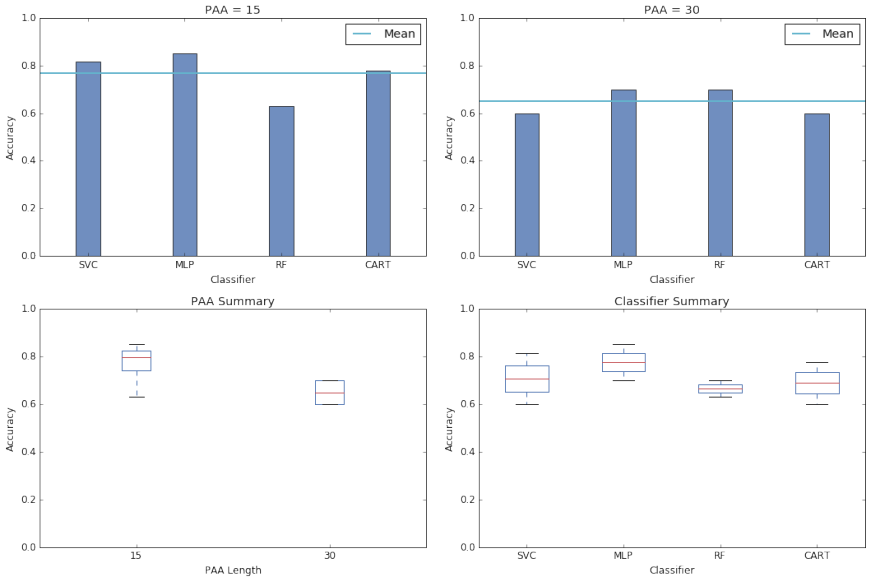


Figure 23: Results for Manual Extraction and Selection, 1Hz, PCA

6.2.5 Summary

The best classifier that could be built using manual extraction and selection with full SCPD as feature engineering method showed to be an MLP in combination with Setup 1 Hz PCA (PCA, 1Hz sampling rate, PAA length 15). Generally, the manual extraction-selection approach with manual turned out to be a quite good feature engineering approach, yielding an accuracy of 85.2% for the best classifier found.

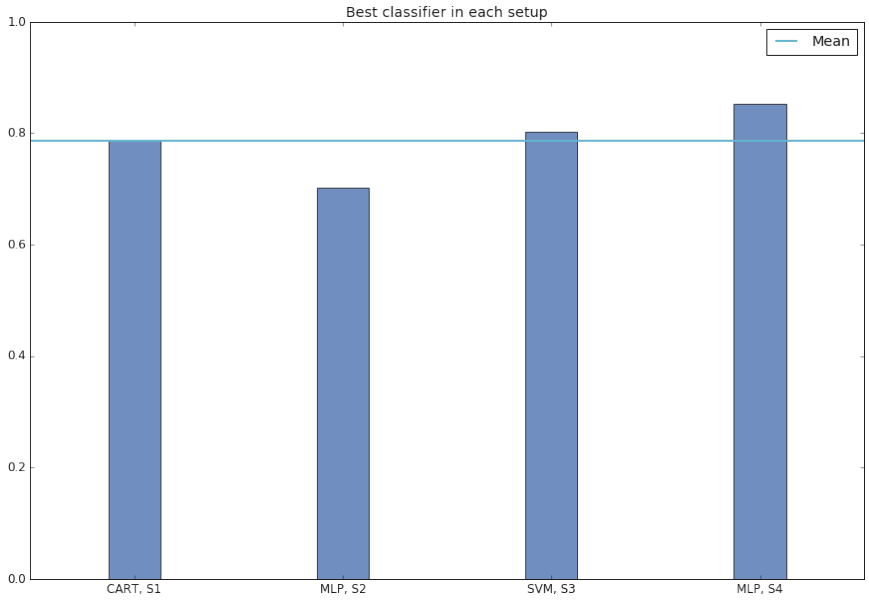


Figure 24: Summary of best classifiers in each setup

6.3 TSFRESH Extraction and Manual Selection with Full SCPD

6.3.1 Setup 10 Hz Euclidean Norm

For Setup 10 Hz Euclidean Norm, features were extracted via TSFRESH and manually selected for all PAA lengths (15, 30, 60, 90, which equals 1.5, 3, 6 and 9 seconds given Setup 10 Hz Euclidean Norms 10Hz sampling rate). All classifiers performed similarly for all PAA lengths with accuracy scores in the upper 50% and lower 60% range except and MLP that achieved and accuracy of 70.1% for a PAA length of 60, which equals 6 seconds for a sampling rate of 10Hz.

Classifier	Best PAA Length	Best Accuracy
MLP	60	70.1%
CART	60	63.8%
RF	30	64.2%
SVM	15	63.1 %

Table 7: Accuracies, TSFRESH Extraction with Manual Selection, Setup 10 Hz Euclidean Norm

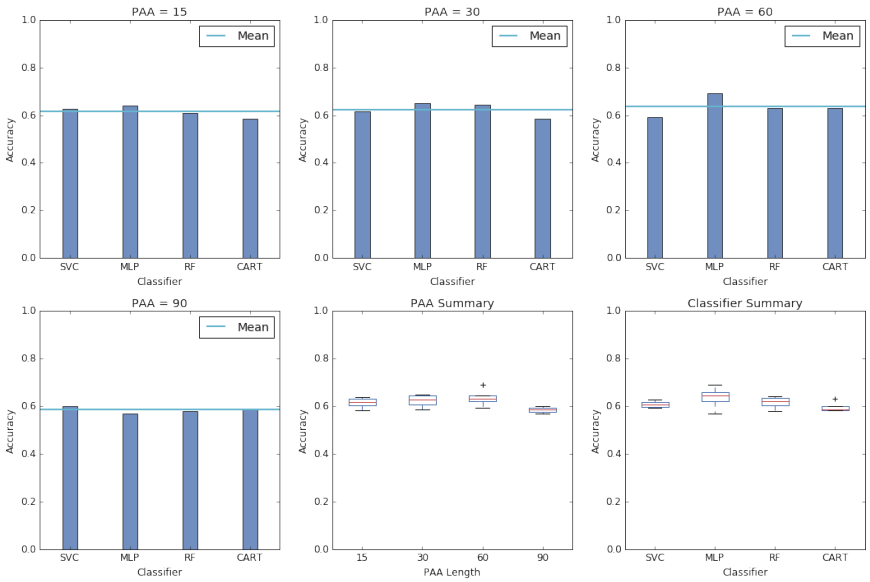


Figure 25: Results for TSFRESH, 10Hz, euclidean norm

6.3.2 Setup 1 Hz Euclidean Norm

For Setup 1 Hz Euclidean Norm, TSFRESH extracted and manually selected features for PAA lengths 15 and 30, which equals 1.5, 3 seconds given Setup 10 Hz Euclidean Norms 10Hz sampling rate). For 60 and 90, no homogeneously labeled subsequences could be found in the the training, test or validation set. All classifiers performed similarly for the respective PAA lengths with accuracy scores in the upper 40% and lower 50% range for PAA length 15 except and in the mid 60% range for a PAA length of 30. Interestingly, the accuracy score the MLP, RF and CART tree turned out to be very similar for a PAA length of 30.

Classifier	Best PAA Length	Best Accuracy
MLP	30	65.0%
CART	30	64.9%
RF	30	65.1%
SVM	30	63.2 %

Table 8: Accuracies, TSFRESH Extraction with Manual Selection, Setup 1 Hz Euclidean Norm

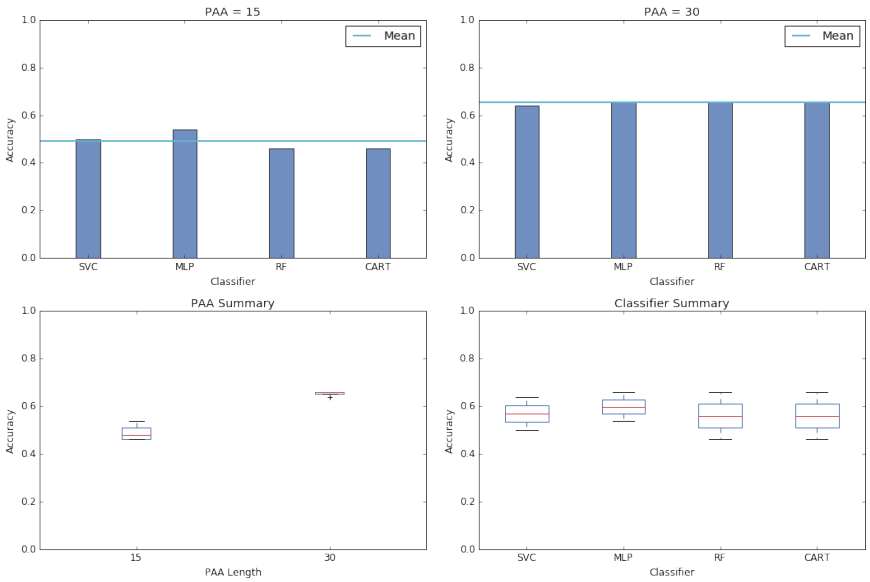


Figure 26: Results for TSFRESH, 1Hz, euclidean norm

6.3.3 Setup 10 Hz PCA

For Setup 10 Hz PCA, TSFRESH extracted and manually selected features for all PAA lengths (15, 30, 60, 90, which equals 1.5, 3, 6 and 9 seconds given Setup 10 Hz Euclidean Norms 10Hz sampling rate). All classifiers performed similarly for all PAA lengths with accuracy scores in the mid to upper 50% and lower 60% range. The best classifier found turned out to be an MLP which achieved a score of 59% for both PAA lengths 60 and 90, which equals 6 and 9 seconds for a 10Hz sampling rate.

Classifier	Best PAA Length	Best Accuracy
MLP	60, 90	59.0%
CART	15	55.4%
RF	30	56.1%
SVM	30	58.8 %

Table 9: Accuracies, TSFRESH Extraction with Manual Selection, Setup 10 Hz PCA

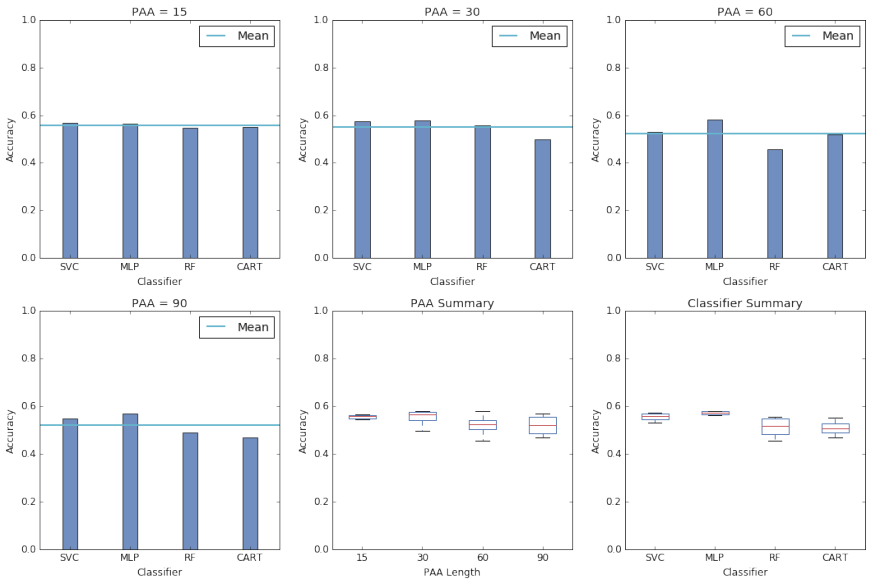


Figure 27: Results for TSFRESH, 10Hz, PCA

6.3.4 Setup 1 Hz PCA

For Setup 1 Hz Euclidean Norm, TSFRESH extracted and manually selected features for PAA lengths 15 and 30, which equals 1.5, 3 seconds given Setup 10 Hz Euclidean Norms 10Hz sampling rate). For 60 and 90, no homogeneously labeled subsequences could be found in the the training, test or validation set. All classifiers performed similarly for the respective PAA lengths with accuracy scores in the upper 40% and lower 50% range for PAA length 15 except and in the mid 60% range for a PAA length of 30. Interestingly, the accuracy score the MLP, RF and CART tree turned out to be very similar for a PAA length of 30.

Classifier	Best PAA Length	Best Accuracy
MLP	30	58.4%
CART	30	58.6%
RF	30	58.1%
SVM	15	53.5 %

Table 10: Accuracies, TSFRESH Extraction with Manual Selection, Setup 1 Hz PCA

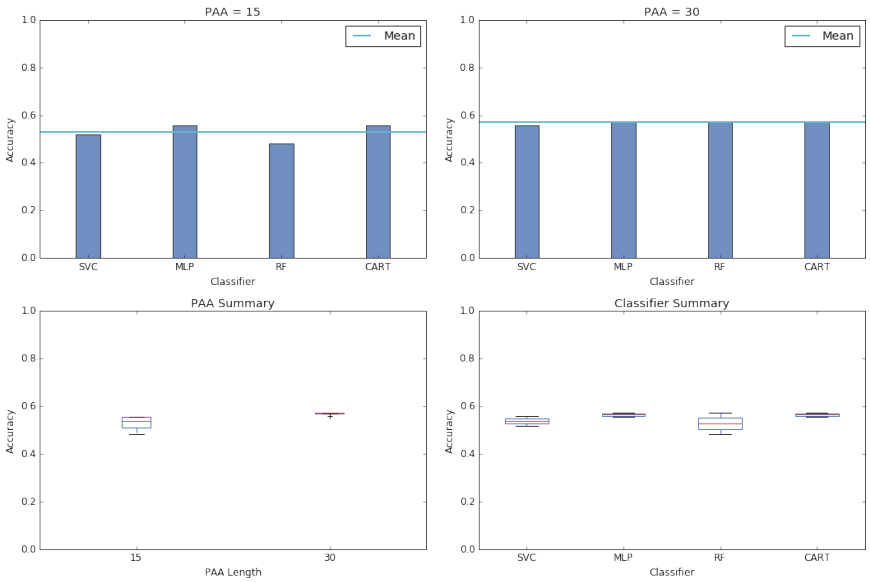


Figure 28: Results for TSFRESH, 1Hz, PCA

6.3.5 Summary

The best classifier that could be built using TSFRESH extraction with manual feature selection as feature engineering method showed to be an MLP in combination with Setup 10 Hz Euclidean Norm (euclidean norm, 10Hz sampling rate, PAA length 60). Generally, the TSFRESH extraction approach with manual feature selection turned out to be not an ideal feature engineering approach, yielding only an accuracy of 70.1% for the best classifier found. Still it outperformed the approach that lets TSFRESH extract and select features using hypothesis testing.

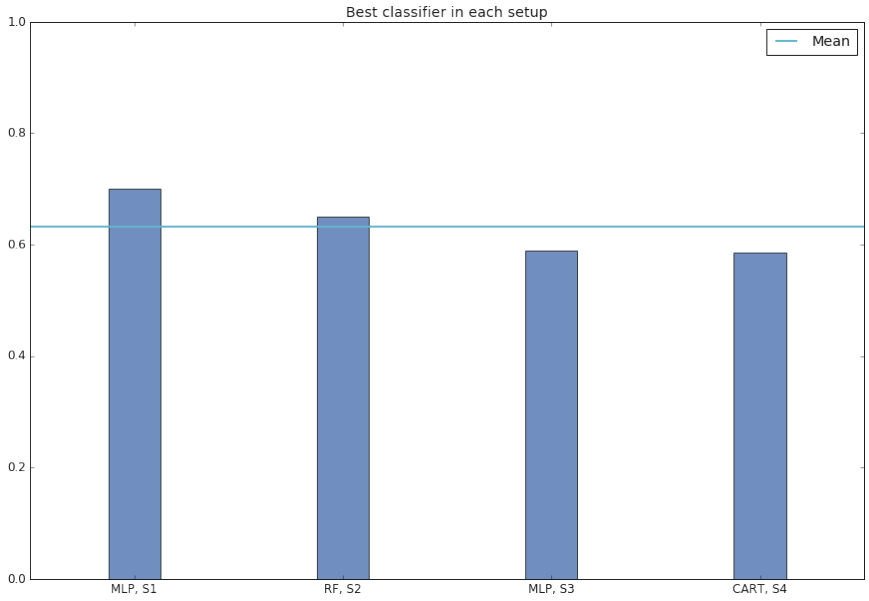


Figure 29: Summary of best classifiers in each setup

6.4 TSFRESH Extraction and Selection with Full SCPD

6.4.1 Setup 10 Hz Euclidean Norm

For Setup 10 Hz Euclidean Norm, TSFRESH extract-selected features for all PAA lengths (15, 30, 60, 90, which equals 1.5, 3, 6 and 9 seconds given Setup 10 Hz Euclidean Norms 10Hz sampling rate). None of the classifiers engineered performed significantly better than random chance one the validation set except the RF classifier. For a PAA length of 60, a RF classifier was found that achieved a 65.2% accuracy score on the validation set.

Classifier	Best PAA Length	Best Accuracy
MLP	15	54.1%
CART	15	59.8%
RF	60	65.2%
SVM	15	58.7 %

Table 11: Accuracies, TSFRESH Extraction and Selection, Setup 10 Hz Euclidean Norm

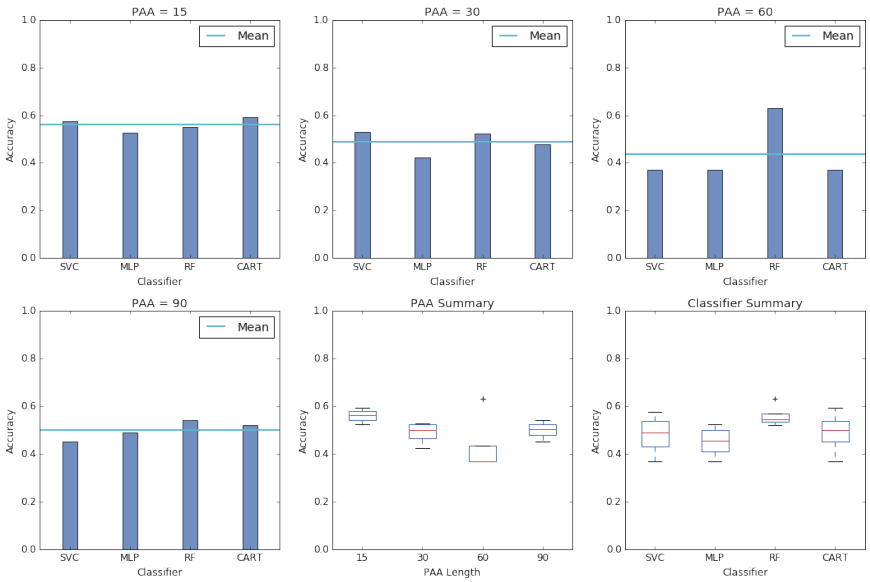


Figure 30: Results for TSFRESH, 10Hz, euclidean norm

6.4.2 Setup 1 Hz Euclidean Norm

For Setup 1 Hz Euclidean Norm, TSFRESH could only extract-select features for a PAA length of 15, which equals 15 seconds at a 1Hz sampling rate. This means that no features sufficiently discriminative could be found for the specified FDR level for PAA lengths 30, 60 and 90. The classification accuracy of the various classifiers was mostly determined by class distribution of the validation set, which indicates that using the features extracted in Setup 1 Hz Euclidean Norm, none of the evaluated classifiers could yield a result better than random chance.

Classifier	Best PAA Length	Best Accuracy
MLP	15	53.8%
CART	15	46.1%
RF	15	45.9%
SVM	15	50.6 %

Table 12: Best Accuracies, TSFRESH Extraction and Selection, Setup 1 Hz Euclidean Norm

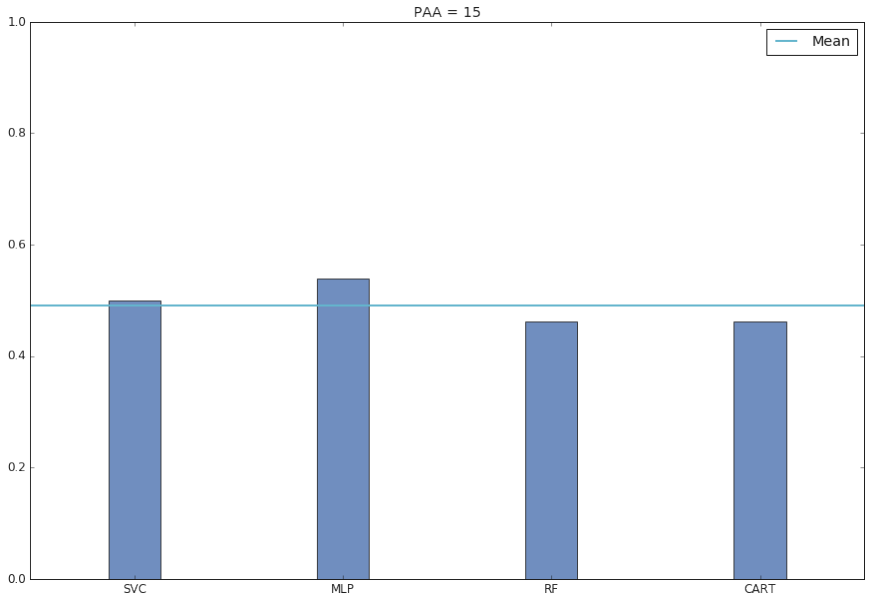


Figure 31: Results for TSFRESH, 1Hz, euclidean norm

6.4.3 Setup 10 Hz PCA

For Setup 10 Hz PCA, TSFRESH extract-selected features for all PAA lengths (15, 30, 60, 90, which equals 1.5, 3, 6 and 9 seconds given Setup 10 Hz Euclidean Norms 10Hz sampling rate). However none of the classifiers engineered performed significantly better than random chance on the validation set.

Classifier	Best PAA Length	Best Accuracy
MLP	15	51.6%
CART	15	54.0%
RF	15	54.0%
SVM	30	50.2 %

Table 13: Best Accuracies, TSFRESH Extraction and Selection, Setup 10 Hz PCA

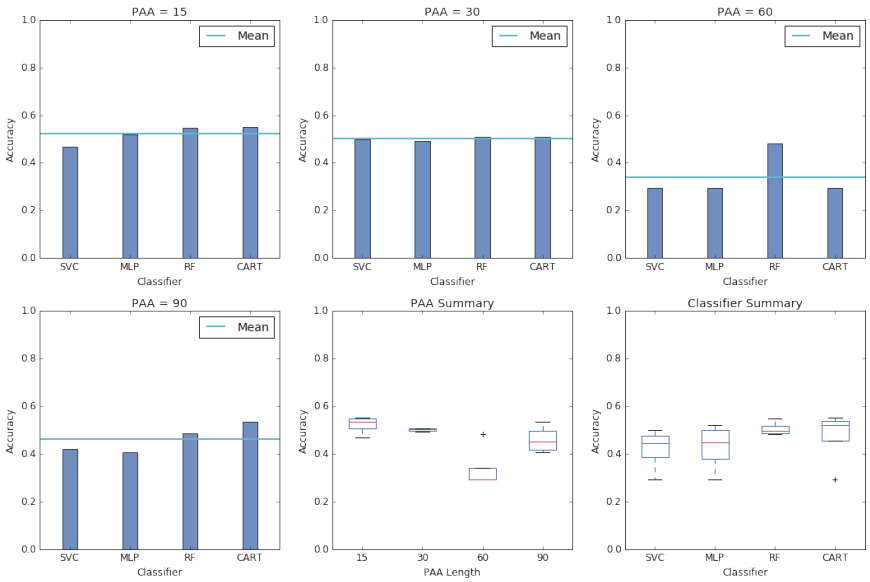


Figure 32: Results for TSFRESH, 10Hz, PCA

6.4.4 Setup 1 Hz PCA

TSFRESH could not extract-select any features from Setup 1 Hz PCA for the given PAA lengths. This means that no features sufficiently discriminative could be found for the specified FDR level.

6.4.5 Summary

The best classifier that could be built using TSFRESH extract-selection as feature engineering method showed to be an RF in combination with Setup 10 Hz Euclidean Norm (euclidean norm, 10Hz sampling rate, PAA length 60). Generally, the TSFRESH extract-selection process turned out to be not an ideal feature engineering approach, yielding only an accuracy of 65.2% for the best classifier found, with most combinations of hyperparameters returning a classifier incapable of classifying a given dataset better than random chance.

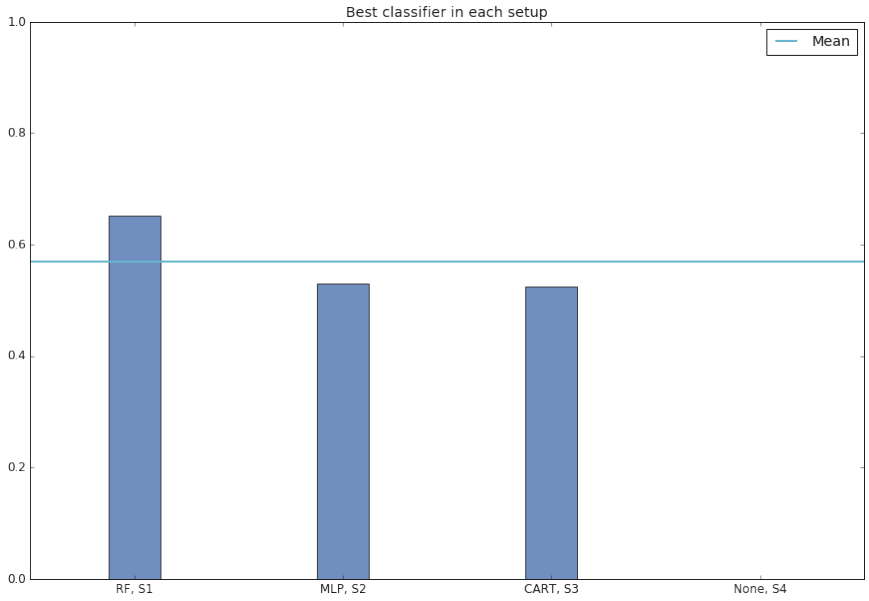


Figure 33: Summary of best classifiers in each setup

6.5 SAX/HIME without SCPD

6.5.1 Summary

For the Sussex-Huawei Locomotion Dataset, SAX/HIME delivered results not better than random chance throughout whole hyperparameter search space. Hence for brevity, only summary statistics of the 48 different combinations of FastSAX window size, FastSAX size and FastSAX alphabet size are presented here as there would be no purpose in going further into the details of the individual experiments. In no experiment, the SAX/HIME approach yielded a classifier with an accuracy score better than 58.8% ().

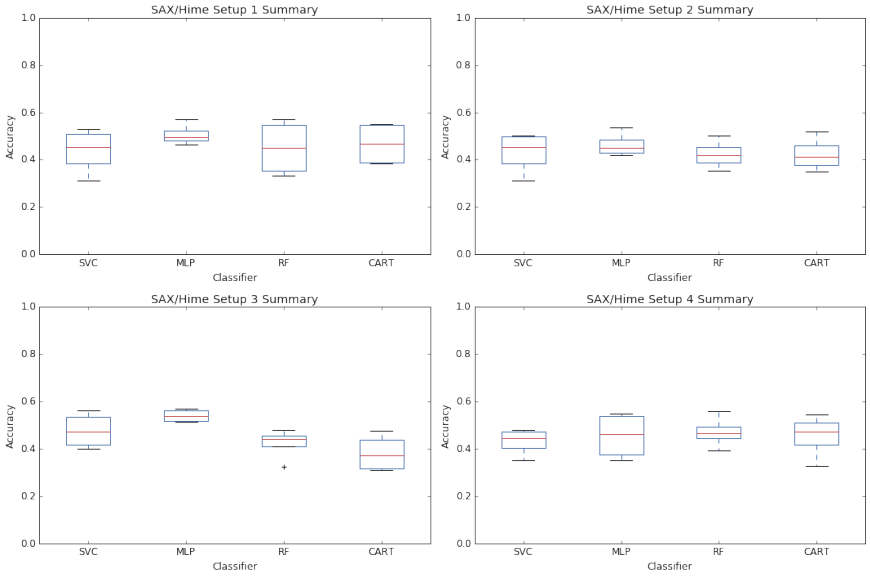


Figure 34: Summary of classifiers in each setup

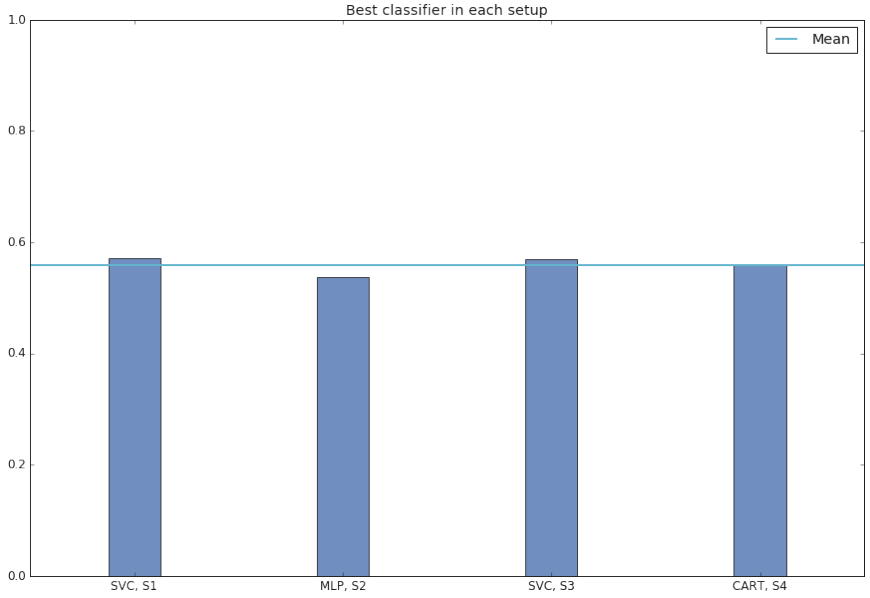


Figure 35: Summary of best classifiers in each setup

6.5.2 Sanity Check

Because of the surprisingly bad results achieved by SAX/HIME on the Sussex-Huawei Locomotion Dataset, we conducted a sanity check to verify that the implementation is working correctly. The experiments for the hyperparameter search space were repeated using dataset 1 of 5 test datasets that come with the example implementation of the grammar-viz wrapper motif-classify, assuming that dataset 1 has already been properly preprocessed. Dataset 1 is an artificially generated 3-class dataset with the following shape:
Train data: 30×128
Validation data: 900×128
The class distribution is given by:

Label 1	Label 2	Label 3
33.4%	33.0%	33.6%

Table 14: Dataset 1, class distribution

The sanity check using dataset 1 yielded significantly better results than the hyperparameter search on the Sussex Huawei Locomotion Dataset, which verifies the correctness of the implementation in principle and implies that this feature engineering approach in its current form does not yield acionable result for the Sussex Huawei Locomotion Dataset.

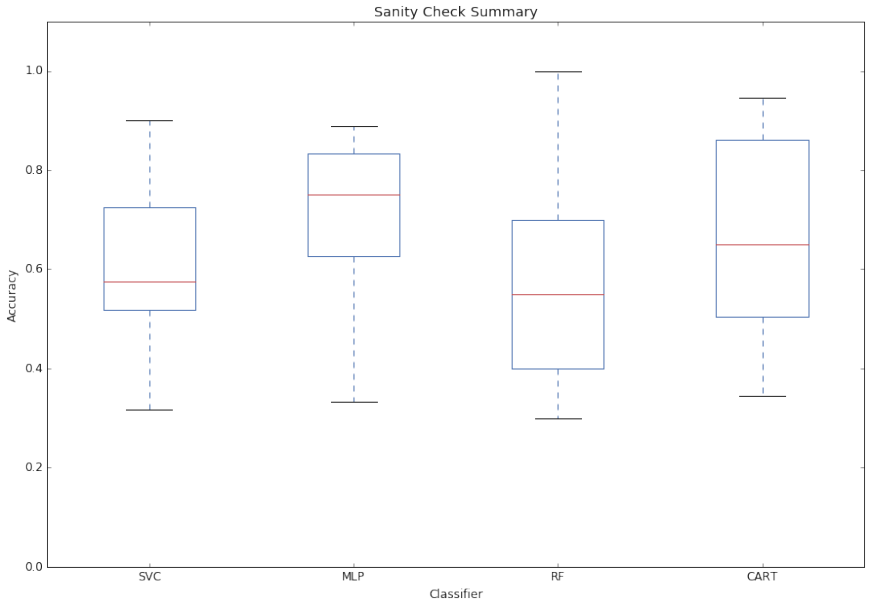


Figure 36: Summary of best classifiers in each setup

6.6 MP with Semi-SCPD

The experiments regarding MPs were conducted using semi-SCPD. In semi-SCPD, SCPD is only applied to the training set, while test and validation sets remain untouched.

6.6.1 Setup 10 Hz Euclidean Norm

For Setup 10 Hz Euclidean Norm, the best classifier found was an MLP that achieved an accuracy score of 85.2% on the validation set, using features engineered based on motifs extracted via SCRIMP++ with a motif radius of 10 and an a motif length of 150. Given the 10Hz resampling rate using in Setup 10 Hz Euclidean Norm, this means that the optimal motif length found for Setup 10 Hz Euclidean Norm is 15 seconds long and that each two sub-sequences where the motif was found are at least 1 second apart. Both classes were roughly equally well classified, with "City" trips beeing classified with an accuracy of 80.0% and "Country Road" trips beeing classified with an accuracy of 84.6%.

Classifier	Best Motif Length	Best Motif Radius	Best Accuracy
MLP	150	10	85.2%
CART	200	100	78.6%
RF	150	200	77.3%
SVM	100	200	79.4%

Table 15: Accuracies, Matrix Profiles, Setup 10 Hz Euclidean Norm

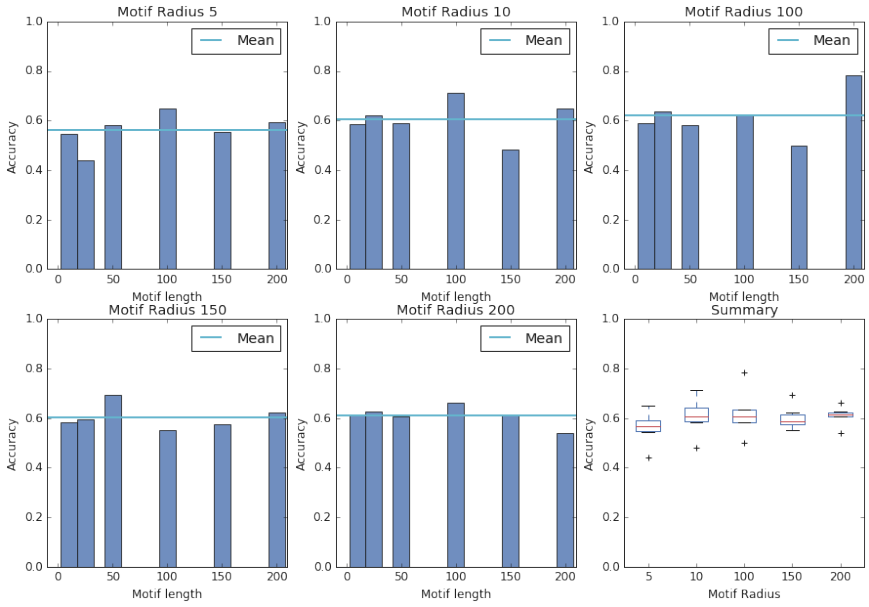


Figure 37: Results for CART, 10Hz, euclidean norm

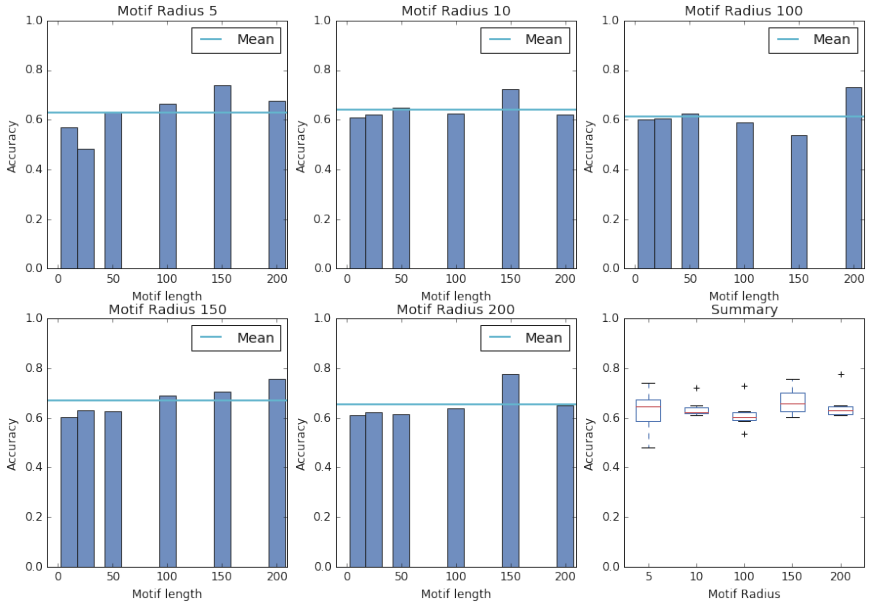


Figure 38: Results for RF, 10Hz, euclidean norm

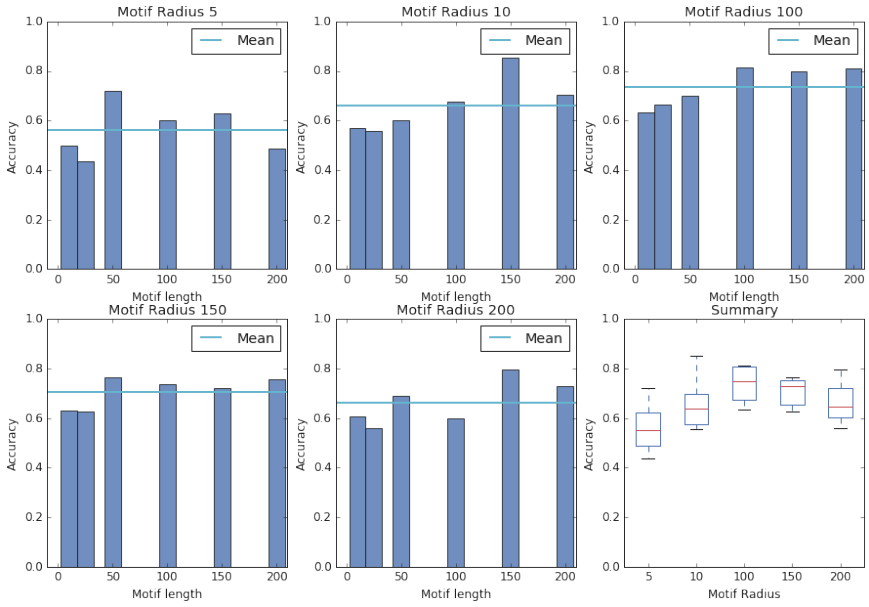


Figure 39: Results for MLP, 10Hz, euclidean norm

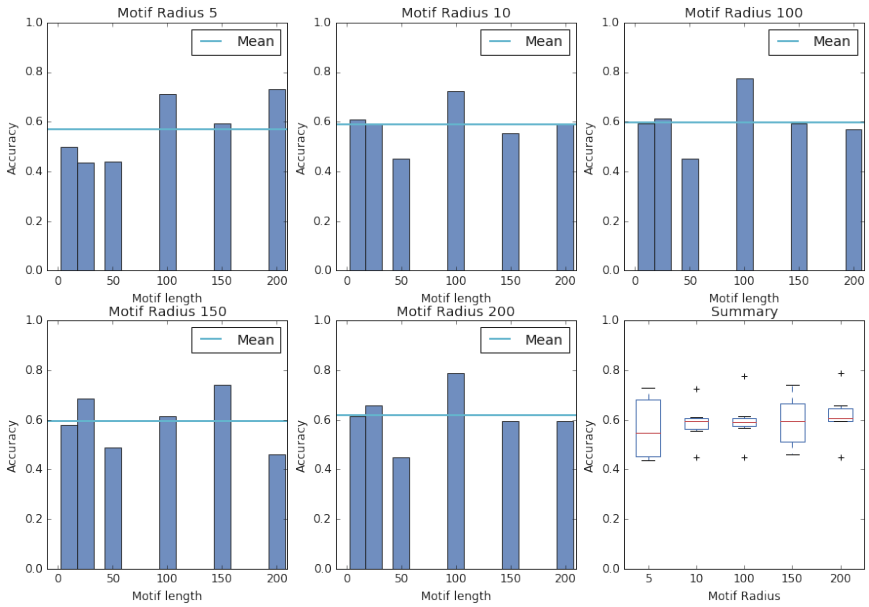


Figure 40: Results for SVM, 10Hz, euclidean norm

6.6.2 Setup 1 Hz Euclidean Norm

For Setup 1 Hz Euclidean Norm, the best classifier found was a CART tree that achieved an accuracy score of 70.8% on the validation set, using features engineered based on motifs extracted via SCRIMP++ with a motif radius of 200 and an a motif length of 5. Given the 1Hz resampling rate using in Setup 1 Hz Euclidean Norm, this means that the optimal motif length found for Setup 10 Hz Euclidean Norm is 5 seconds long and that each two sub-sequences where the motif was found are at least 200 seconds apart. One class was distinctively preferred by the classifier, with "City" trips beeing classified with an accuracy of 54.2% and "Country Road" trips beeing classified with an accuracy of 86.5%.

Classifier	Best Motif Length	Best Motif Radius	Best Accuracy
MLP	100	150	67.1%
CART	10	200	70.8%
RF	100	200	68.3%
SVM	25	10	66.9%

Table 16: Accuracies, Matrix Profiles, Setup 1 Hz Euclidean Norm

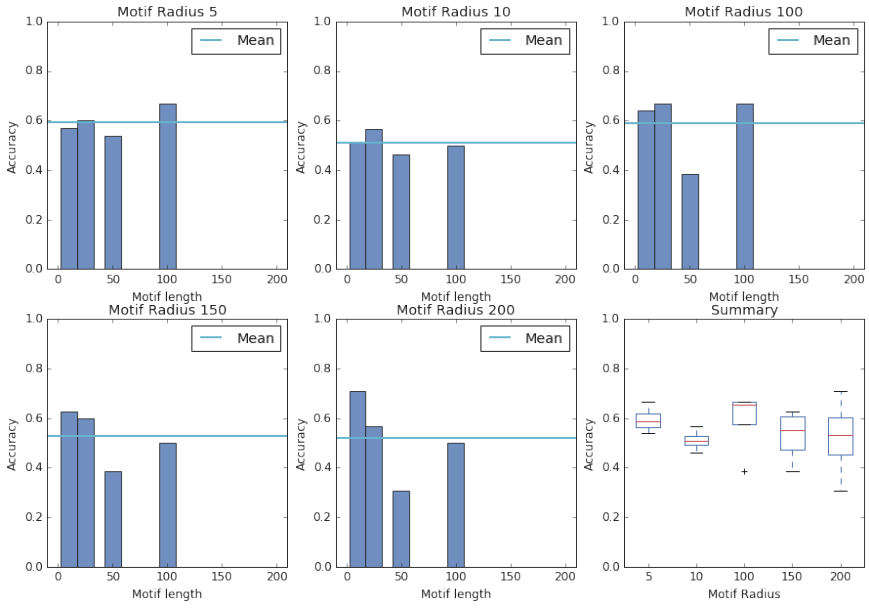


Figure 41: Results for CART, 1Hz, euclidean norm

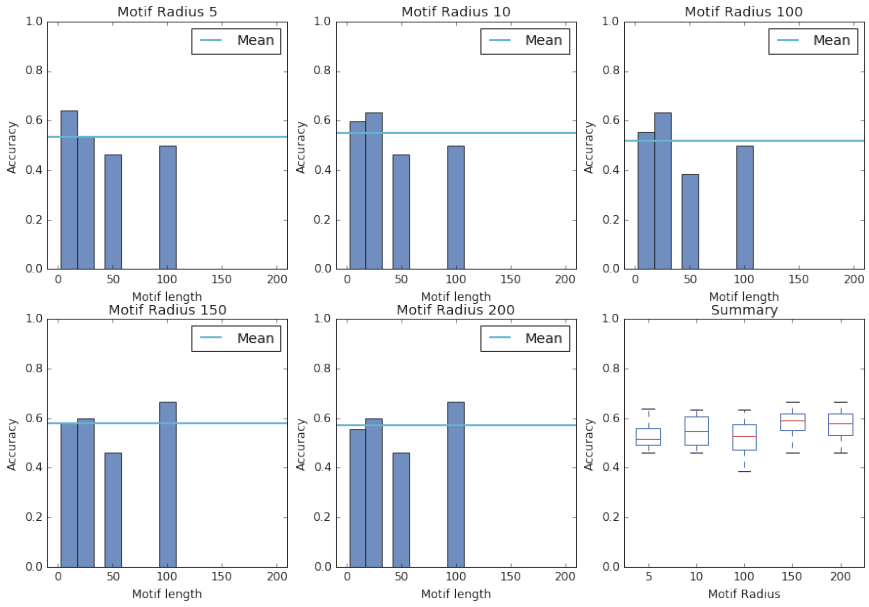


Figure 42: Results for RF, 1Hz, euclidean norm

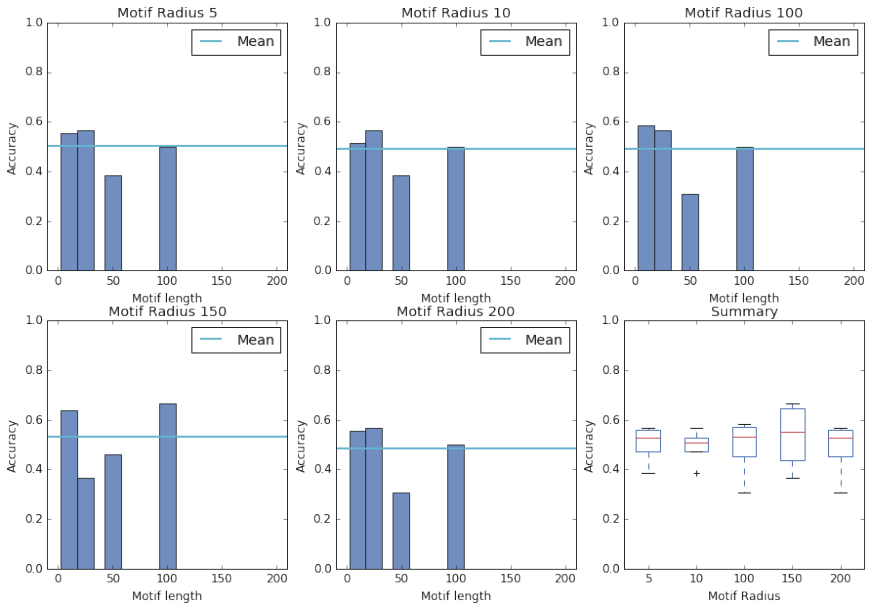


Figure 43: Results for MLP, 1Hz, euclidean norm

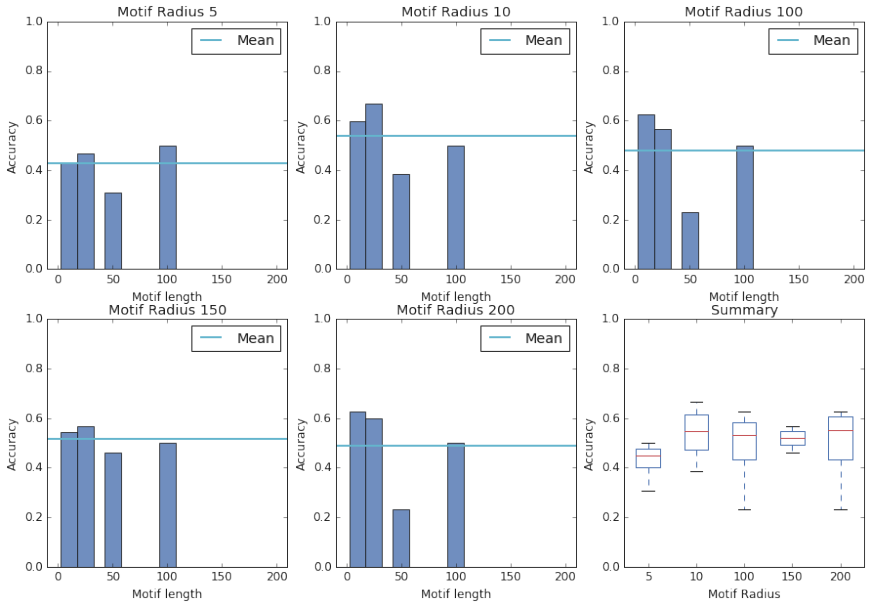


Figure 44: Results for SVM, 1Hz, euclidean norm

6.6.3 Setup 10 Hz PCA

For Setup 10 Hz PCA, the best classifier found was an MLP that achieved an accuracy score of 69.1% on the validation set, using features engineered based on motifs extracted via SCRIMP++ with a motif radius of 150 and an a motif length of 100. Given the 10Hz resampling rate using in Setup 10 Hz PCA, this means that the optimal motif length found for Setup 10 Hz Euclidean Norm is 10 seconds long and that each two sub-sequences where the motif was found are at least 15 second apart. One class was distinctively preferred by the classifier, with "City" trips beeing classified with an accuracy of 40.0% and "Country Road" trips beeing classified with an accuracy of 91.3%.

Classifier	Best Motif Length	Best Motif Radius	Best Accuracy
MLP	100	150	69.1%
CART	100	10	59.8%
RF	10	100	57.5%
SVM	25	150	60.7%

Table 17: Accuracies, Matrix Profiles, Setup 10 Hz PCA

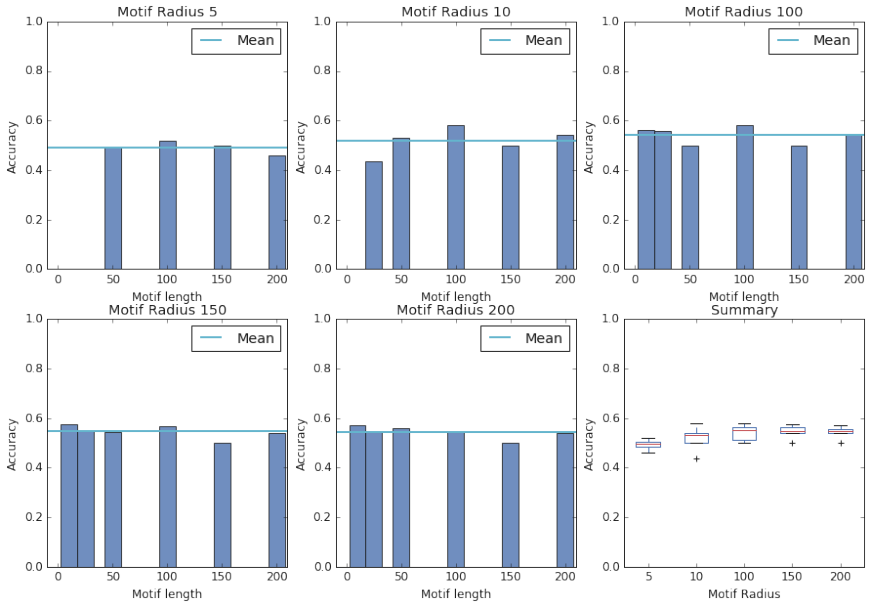


Figure 45: Results for CART, 10Hz, PCA

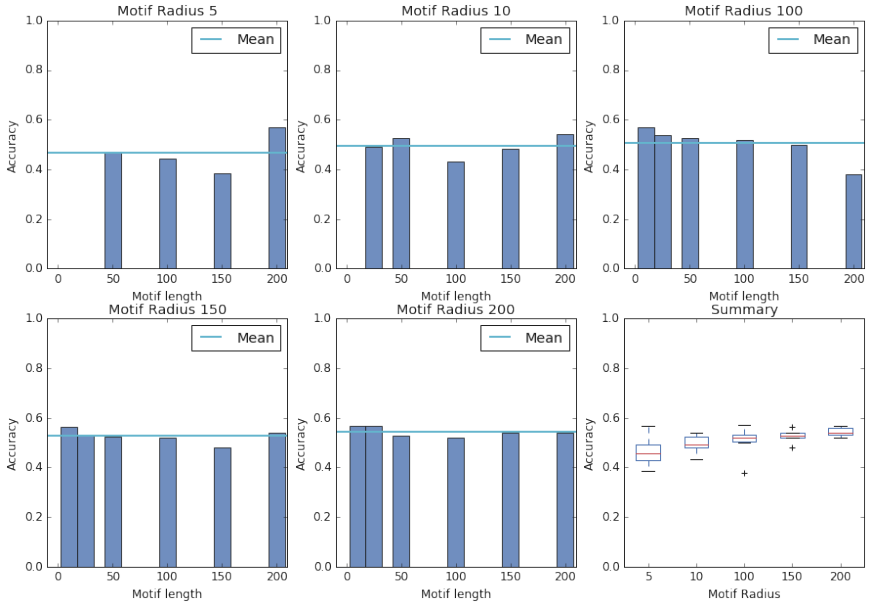


Figure 46: Results for RF, 10Hz, PCA

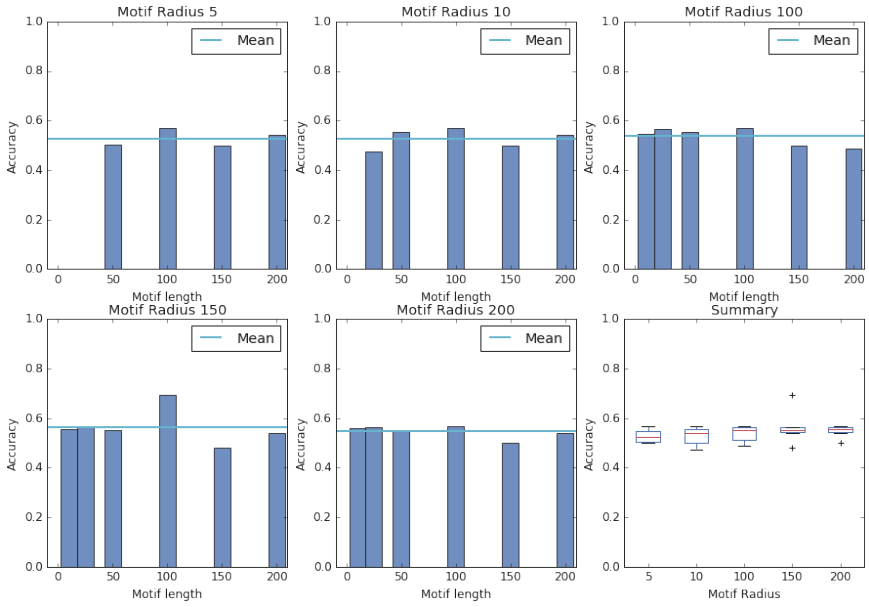


Figure 47: Results for MLP, 10Hz, PCA

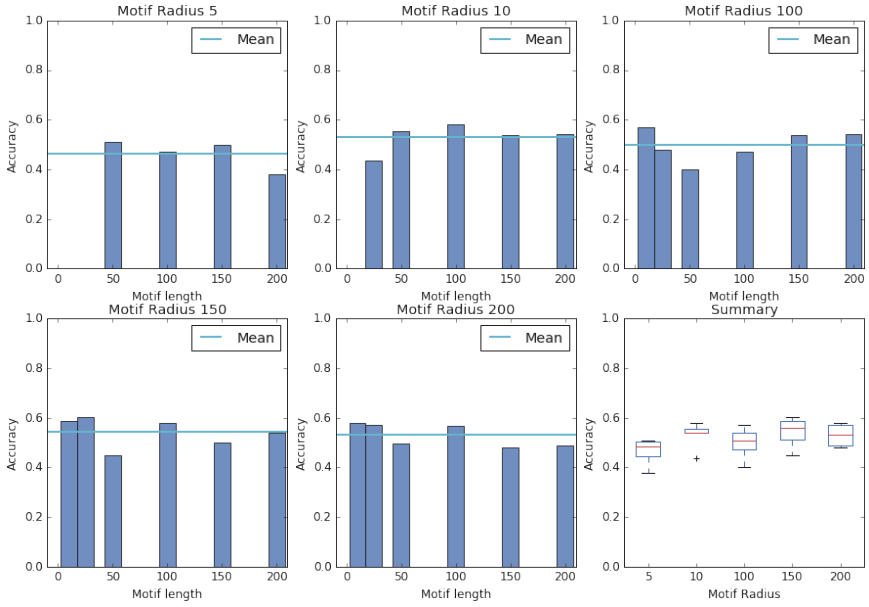


Figure 48: Results for SVM, 10Hz, PCA

6.6.4 Setup 1 Hz PCA

For Setup 1 Hz PCA, the best classifier found was a CART tree that achieved an accuracy score of 73.3% on the validation set, using features engineered based on motifs extracted via SCRIMP++ with a motif radius of 10 and an a motif length of 50. Given the 1Hz resampling rate using in Setup 1 Hz PCA, this means that the optimal motif length found for Setup 10 Hz Euclidean Norm is 10 seconds long and that each two sub-sequences where the motif was found are at least 50 second apart. One class was distinctively preferred by the classifier, with "City" trips beeing classified with an accuracy of 66.7% and "Country Road" trips beeing classified with an accuracy of 77.7%.

Classifier	Best Motif Length	Best Motif Radius	Best Accuracy
MLP	100	200	66.8%
CART	50	10	73.3%
RF	25	150	61.2%
SVM	50	10	60.7%

Table 18: Accuracies, Matrix Profiles, Setup 1 Hz PCA

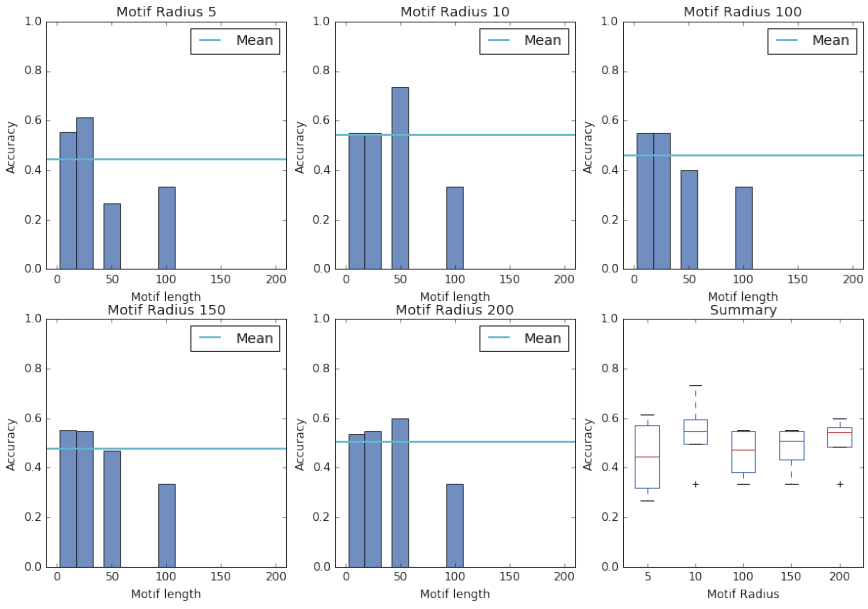


Figure 49: Results for CART, 1Hz, PCA

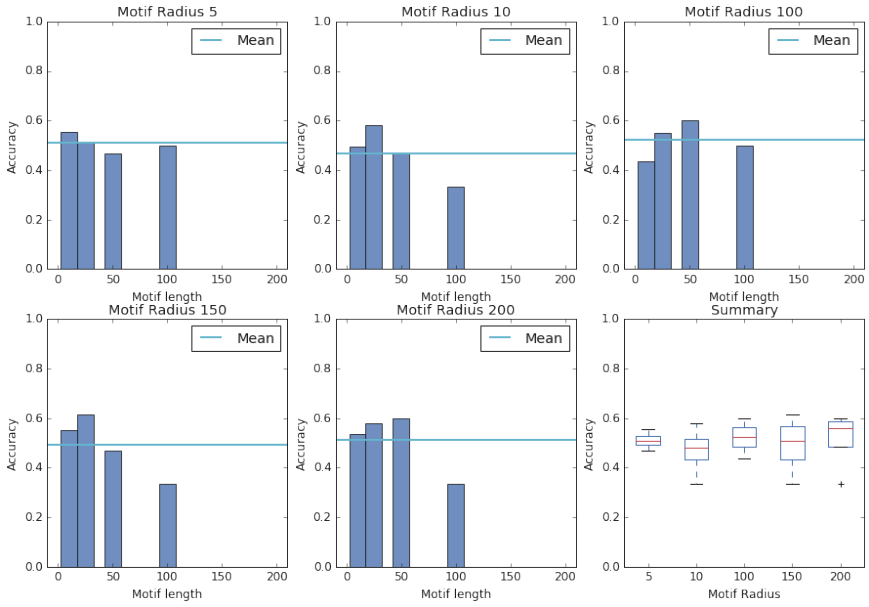


Figure 50: Results for RF, 1Hz, PCA

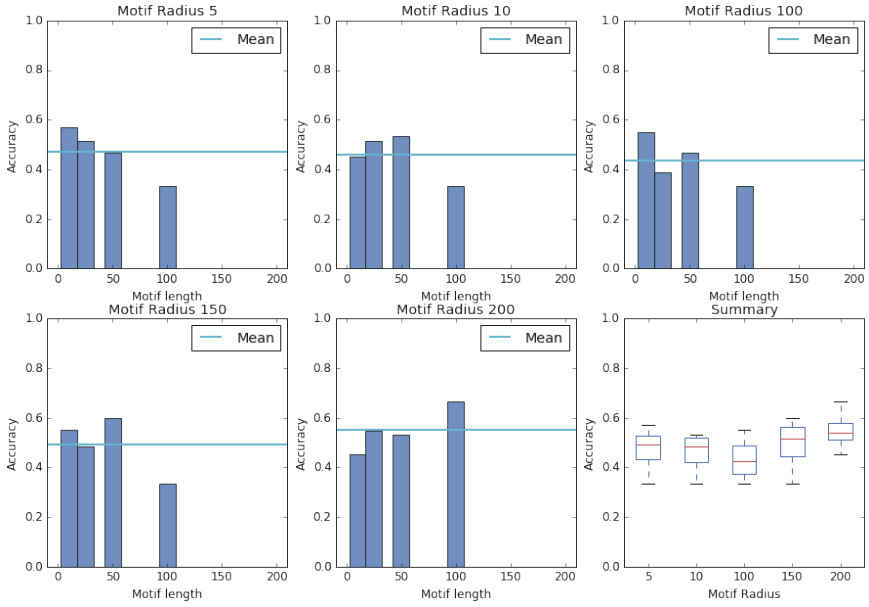


Figure 51: Results for MLP, 1Hz, PCA

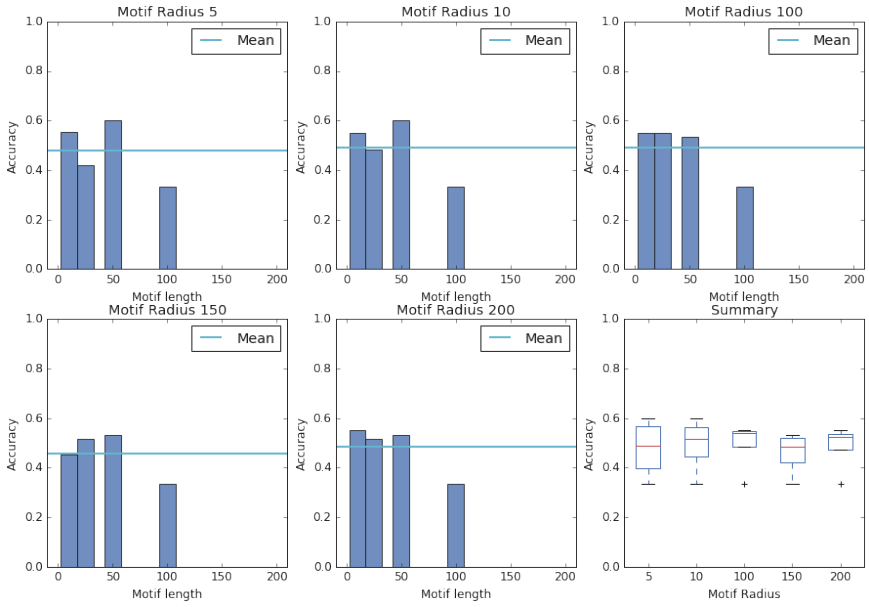


Figure 52: Results for SVM, 1Hz, PCA

6.6.5 Summary

The best classifier that could be built using motif discovery as feature engineering method showed to be an MLP in combination with setups 1 (euclidean norm, 10Hz sampling rate, 15 seconds motif length and 1 second motif radius) that achieved an accuracy score of 85.2%. Interestingly, the optimal classifier for features extracted by motif discovery depends on the sampling rate: for motifs extracted from a time series with a sampling rate of 1Hz, CART always turned out to be optimal, while for the experiments conducted with 10Hz sampling rate, the MLP achieved the highest accuracy score.

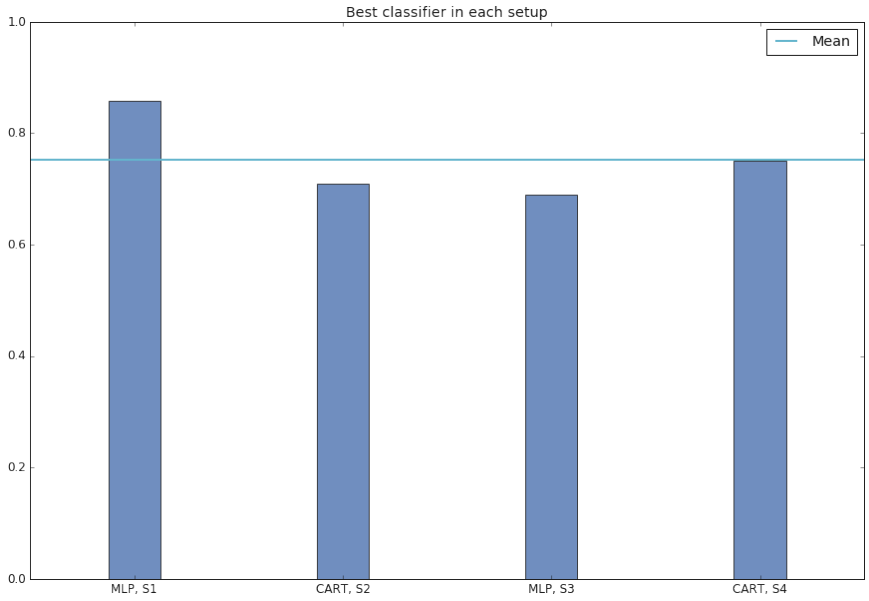


Figure 53: Summary of best classifiers in each setup

7 Conclusion

7.1 Statistical Features

Statistical feature extraction and selection turned out to be a viable approach, but generally significantly dependent on the CPD problem. As shown in experiments regarding statistically extracted and selected features, even the approach that performed best, manual extraction of a preselected set of features, proofed to be highly sensitive to weather or not the segments approximated with PAA are homogeneously labeled or not (see sections 6.2 and 6.2), with the best accuracy score achieved by any classifier without SCPD beeing 79.0% and the best accuracy achieved with full SCPD beeing 85.2%. Interestingly, TSFRESH performed relatively bad compared to the manual statistical feature extraction and selection approach, which is an unexpected result. The experiments described in sections 6.3 and 6.4 showed that in the case where TSFRESH extracts and selects features with full SCPD based on statistical hypothesis testing, the best accuracy score achieved by any classifier was 65.2%, and the accuracy achieved by any classifier in the case where TSFRESH with SCPD extracts a predefined set of manually selected features was 70.1%.

7.2 Motifs as Features

The experiments regarding the suitability of motifs as features for the given classification task yielded many actionable results. It turned out that, as shown in section 6.6, that the motifs engineered based on MPs extracted via the SCRIMP++ algorithm could be aggregated using tagging and summary statistics in such a way that they become highly discriminative features, resulting in a best classification accuracy of 85.2%. While this as exactly as good as the accuracy achieved using features engineered by manual extraction and selection with SCPD, it has to be noted that only Semi-SCPD was used during the motif feature engineering process, which indicates a higher robustness of motif based features against the CPD problem when compared to statistically engineered features. Surprisingly, the features engineered based on the SAX/HIME approach yielded only in classifiers that were not much better than random chance (see section 6.5), with the accuracy score beeing mostly defines by the class distribution in the dataset. This is unexpected because during sanity checks conducted by applying the SAX/HIME approach on computer generated toy-datasets instead of the Sussex Huawei Locomotion dataset, the classification accuracy was very good, with accuracy score in the 80 to 90% range.

7.3 Classifiers

Of the classifiers that were evaluated in this work, the MLP and the CART tree were selected as best classifiers in 10 respectively 9 cases in any setup of any feature extraction method. The SVC turned out to be the best classifier in 5 cases and the RF in 3 cases respectively. This indicates that an MLP and a CART tree are best suited for problems of this kind, independent of the feature extraction method used. Interestingly, for both the statistically engineered as well as the MP based motifs, the best classifier turned out to be an MLP with a 85.2% accuracy score.

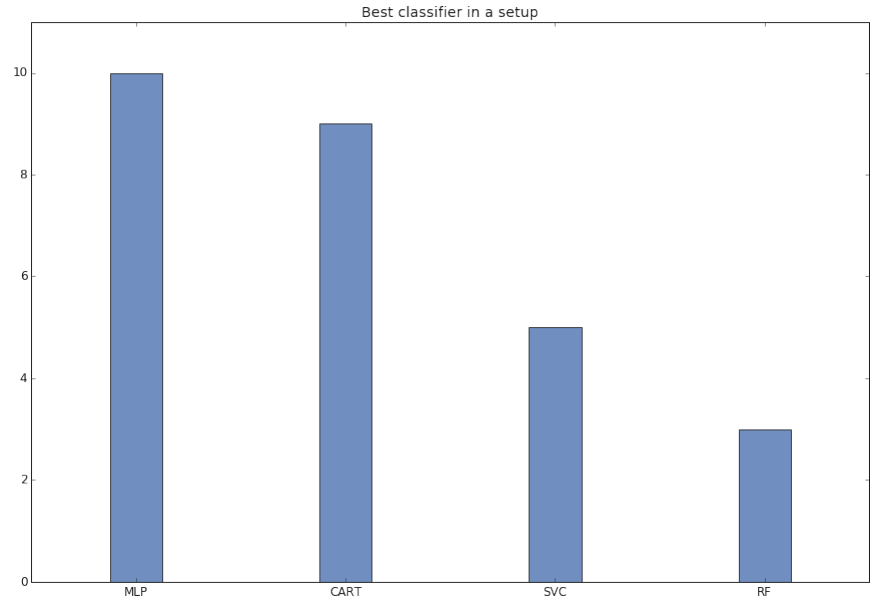


Figure 54: How many times a classifier was chosen as best classifier in a setup.

7.4 PAA lengths

7.4.1 Motifs via MPs

For motifs, the best classifier (MLP, 85.2% accuracy on validation set) used a 15 seconds motif length and generally, the two most common motif lengths used by the best classifier in each setup were 10 seconds and 15 seconds. This indicates that the most discriminative patterns that can be discovered in telemetry data via the MP motif discovery approach are in the 10 to 15 seconds range.

Motif length (seconds)	1.5	2.5	5.0	10.0	15.0	20.0
Used by best classifier in any setup	0	0	1	2	2	0

Table 19: Motif lengths

Motif length (seconds)	25	50	100	150	200
Used by best classifier in any setup	0	0	0	0	0

Table 20: Motif lengths

7.4.2 Statistically Engineered Features

For statistically engineered features, the best classifier (MLP again, 85.2% accuracy on validation set) used a 15 seconds motif length while generally, the

three most common PAA lengths used by the best classifier in each setup were 30, 3 and 6 seconds and 15 seconds. This indicates that the most discriminative patterns that can be discovered in telemetry data via the statistical feature extraction approach are in the 3 to 30 seconds range.

PAA length (seconds)	1.5	3.0	6.0	9.0
Used by best classifier in any setup	0	3	3	2

Table 21: PAA lengths

PAA length (seconds)	15	30	60	90
Used by best classifier in any setup	2	4	0	0

Table 22: PAA lengths

7.5 Dimensionality Reduction Techniques

It turned out that in 4 out of 5 experiments, the best classifier in any setup used the euclidean dimensionality reduction technique, while PCA only once was used by the best classifier. This indicates a clear preference by the classifiers for dimensionality reduction via euclidean norm.

Dimensionality Reduction	PCA	Euclidean
Used by best classifier in any experiment	1	4

Table 23: PAA lengths

7.6 Impact of SCPD

To determine the impact of SCPD on the statistical feature extraction techniques, the experiment for manual extraction and selection was carried out with and without SCPD and the accuracy of the bets classifier was 6.2% better in the case where SCPD was applied. This indicates that CPD is a relevant problem for statistical feature extraction methods and further research should be done in this area.

SCPD	Yes	No
Best Accuracy	85.2%	79.0%

Table 24: PAA lengths

8 Future Work and Outlook

8.1 Using Sophisticated Neural Networks

The best classifier found in this work was an MLP for both the statistical features and the features engineered via MPs. This would make researching the effects of MLP architecture choices on the accuracy as well as using other more sophisticated neural network types (CNNs, RNNs) an interesting topic.

8.2 In-Depth Evaluation of TSFRESHs Performance

This work showed that the features engineered via TSFRESH, despite it's highly sophisticated and intuitively better approach, performed worse than the manually extracting simple statistical features. This leads to the question why TSFRESH performed so badly and if using different configurations (hyperparameter choice) of TSFRESH would perform better.

8.3 Replace SCPD with CPD

This work only examined the impact of SCPD on on experiment, namely manual feature extraction and selection. For completeness, it would be necessary to examine the impact of SCPD on all experiments.

Furthermore, an interesting research topic would be how using actual CPD algorithms, such as semantic segmentation [23], would affect classification accuracy.

8.4 Other Feature Extraction Methods

This work only covers a tiny fraction of possible feature engineering approaches for time series classification. The following approaches appeared feasible during the initial round of literature research and could be worthy of further research:

- Shapelets are time series sub-sequences which are maximally representative of a class. Intuitively, shapelets are found by evaluating the prediction qualities of numerous candidates extracted from the series segment. [61, 25]
- Instead of using a feature engineering technique that produces features for which the IID assumption holds, classifiers suited for time series data that can generally not assumed to be IID could be used [22]

8.5 Motif Discovery

With specific regard to motif discovery, the approach chosen in this work could be further refined using the following techniques:

- Motion Graphs: *"Motion motifs represent clusters of similar motions and together with their encompassing motion graph they lend understandable structure to the contents and connectivity of large motion datasets. [...] The processed result is multi-purpose, potentially usable for motion resequencing, creating blend spaces, motion compression, motion segmentation, and motion annotation."* [6]

- N-Dimensional-Motif discovery, as described in [62], could be used to remove the constraint of this work that the 3-dimensional accelerometer time series has to be reduced to a 1-dimensional representation, which could result in motifs that are better suited for the classification task because information about the direction of the acceleration is better maintained.

- Guided motif discovery by incorporating domain knowledge into matrix profiles using so called annotation vectors that provide a bias for the relevance of a matrix profile. [16]

- Variable length motif discovery using VALMOD, a matrix profile based algorithm that detects motifs without having to specify a range of lengths to be searched. [33]

8.6 Dataset

Furthermore, only a 25% sample subset of the Sussex Huawei Locomotion data set was used during the authors experiments due to runtime and memory constraints. On more sophisticated hardware, the full data set could be used, which could lead to improved results.

A Algorithms

A.1 SCRIMP++ Pseudo Code

Algorithm 2: *The PreSCRIMP Algorithm*

Input: *A time series T, a subsequence length m and a sampling interval s.*

Output: *The running matrix profile P and matrix profile index I of T*

```
1  n ← Length(T), P← infs, I← ones // initialization
2  μ, σ ← ComputeMeanStd(T, m)  // precomputation, see [7]
3  for i ← RandPerm( 1 : s : (n-m+1) ) do  //sampling with interval s
4     seq ← T_{i,m}  //obtain a sample subsequence
5     D ← MASS (T, seq)  // evaluate a distance profile, see [5]
6     P, I ← ElementWiseMin (D, P, i)
7     P_i , I_i ← min(D)
8     j ← I_i  // the nearest neighbor of the sample subsequence
9     q ← CalculateDotProduct (P_i, μ_i, σ_i, μ_j, σ_j), q' ← q  // see (1)
10    for k ← 1 to min (s-1, n-m+1- max(i,j) ) do
11       q ← q - t_{i+k-1} t_{j+k-1} + t_{i+k+m-1} t_{j+k+m-1}  // see (2)
12       d ← CalculateDistance (q, μ_{i+k}, σ_{i+k}, μ_{j+k}, σ_{j+k}) // see (1)
13       if d < P_{i+k} do P_{i+k} ← d, I_{i+k} ← j+k end if
14       if d < P_{j+k} do P_{j+k} ← d, I_{j+k} ← i+k end if
15    end for
16    q ← q'
17    for k ← 1 to min(s-1, i-1, j-1) do
18       q ← q - t_{i-k+m} t_{j-k+m} + t_{i-k} t_{j-k}  // see (2)
19       d ← CalculateDistance (q, μ_{i-k}, σ_{i-k}, μ_{j-k}, σ_{j-k}) // see (1)
20       if d < P_{i-k} do P_{i-k} ← d, I_{i-k} ← j-k end if
21       if d < P_{j-k} do P_{j-k} ← d, I_{j-k} ← i-k end if
22    end for
23 end for
24 return P, I
```

Figure 55: PreSCRIMP Pseudo Code [64]

Algorithm 1: ***The SCRIMP Algorithm***

Input: *A time series T and a subsequence length m*
Output: *Matrix profile P and matrix profile index I of T*

```
1  n ← Length(T)
2  μ, σ ← ComputeMeanStd(T, m) // see [7]
3  P ← infs, I ← ones     // initialization
4  Orders ← RandPerm(m/4+1 : n-m+1) // randomize evaluation order
5  for k in Orders          //evaluating diagonals in random order
6     for i ← 1 to n-m+2-k
7        if i=1 do q ← DotProduct(T_{1,m}, T_{k,m})
8        else q ← q - t_{i-1} t_{i+k-2} + t_{i+m-1} t_{i+k+m-2}    // see (2)
9        end if
10       d ← CalculateDistance(q, μ_i, σ_i, μ_{i+k-1}, σ_{i+k-1}) // see (1)
11       if d < P_i do P_i ← d, I_i ← i+k-1 end if
12       if d < P_{i+k-1} do P_{i+k-1} ← d, I_{i+k-1} ← i end if
13    end for
14 end for
15 return P, I
```

Figure 56: SCRIMP Pseudo Code [64]

A.2 HIME Pseudo Code

Algorithm 2 Hierarchical based Motif Enumeration (HIME)

Input: Induction Graph G
Parameter: PAA size w, Alphabeta Size a
Output: Motif Set $MotifSet$
VLSAXTable[SAX word, Length][Location]={}
for each node S_i from left to right **do**
    {Compute SAX word $word$ for long subsequence}
  SS=Merge(S_i,$S_i^{forward}$); $word$=FastSAX(SS,w,a);
    {Check if the same SAX word representing some subsequence with similar length is recorded}
  if !VLSAXTable.exist($word$,SS.Length) **then**
    VLSAXTable.put($word$,SS.Length,SS.Location);
  else
    {Retrieve subsequence with matching SAX word}
    $SS2$=VLSAXTable.getSimLengthSeq($word$);
    {Update Induction Graph G}
    InsertMotifNode(G,$SS2$,SS);
    UpdateMotifSet(MotifSet,$word$);
    {Greedy Removing Covered Motifs}
    RemoveCoveredMotif(SS,S_i);
    {Enumerate Motifs based on SS and $SS2$}
    $SS3$=Merge(SS,$SS^{forward}$);
    $SS4$=Merge($SS2$,$SS2^{forward}$);
    RecursiveEnumeration($SS3$);
    RecursiveEnumeration($SS4$);
  end if
end for
return RemoveTrivialAndFalsePostive(MotifSet);

Figure 57: HIME Pseudo Code [19]

Algorithm 3 Determining Alphabet Size for HIME

1: **Input**: Time Series T
2: **Output**: Alphabet size $\bar{a}$
3: **while** $\bar{a}$ not converged **do**
4: Random Sampling S_1,S_2 from time series T
5: a_i=BinarySearchResolution(S_1,S_2);
6: $\bar{a}$=Average(a);
7: **end while**
8: **return** $\bar{a}$

Figure 58: HIME Alphabet Determination Pseudo Code [19]

A.3 Fast SAX Pseudo Code

Algorithm 1 Fast SAX Computation (FastSAX)

1: **Input**: M_x,M_{xx}, PAA size w, subsequence $S_{p,q}$
2: **Output**: PAA representation paa
3: $Ex = M_x(q) - M_x(p)$, $E_{xx} = M_{xx}(q) - M_{xx}(p)$
4: $n = q - p + 1$, $\mu_x = \frac{E_x}{n}$, $\sigma_x = \sqrt{\frac{E_{xx} - E_x^2/n}{n-1}}$
5: **for** every PAA segment **do**
6: $paa_i = (\frac{M_x(paa_{i,end}) - M_x(paa_{i,start})}{n/w} - \mu_x)/\sigma_x$
7: **end for**
8: **return** ConvertToSAX(paa)

Figure 59: FastSAX Pseudo Code [19]

B Lessons Learned

B.1 Experimentation and Scientific writing

During the development of the software package, we conducted numerous experiments in order to determine which configurations and approaches work and which do not. We ran most of these experiments without keeping in mind that we could later use them for writing my bachelors thesis so we did not ensure that they were ran under comparable circumstances, making them worthless for drawing conclusions beyond which approach is technically feasible and which is not. We had to re-run most experiments after finishing development under identical circumstances to ensure my results are comparable and actionable.

With regards to scientific writing, we learned that having a large number of sources requires a management tool such as Mendeley [41] in order to maintain an overview.

B.2 Software Engineering

We only realized very late during the project that Sklearn already offers an implementation of the pipeline pattern [52]. Using one of the usually great Sklearn implementations instead of implementing the pipeline pattern ourselves would certainly have saved us some time.

Another lesson we learned is that it really is a good idea "to write your thesis while your are implementing it". We ended up having more work then necessary because actively writing the thesis gave me much deeper understanding than just passively reading the papers.

C Implementation

C.1 Software Engineering Artefacts

UML Class Diagram

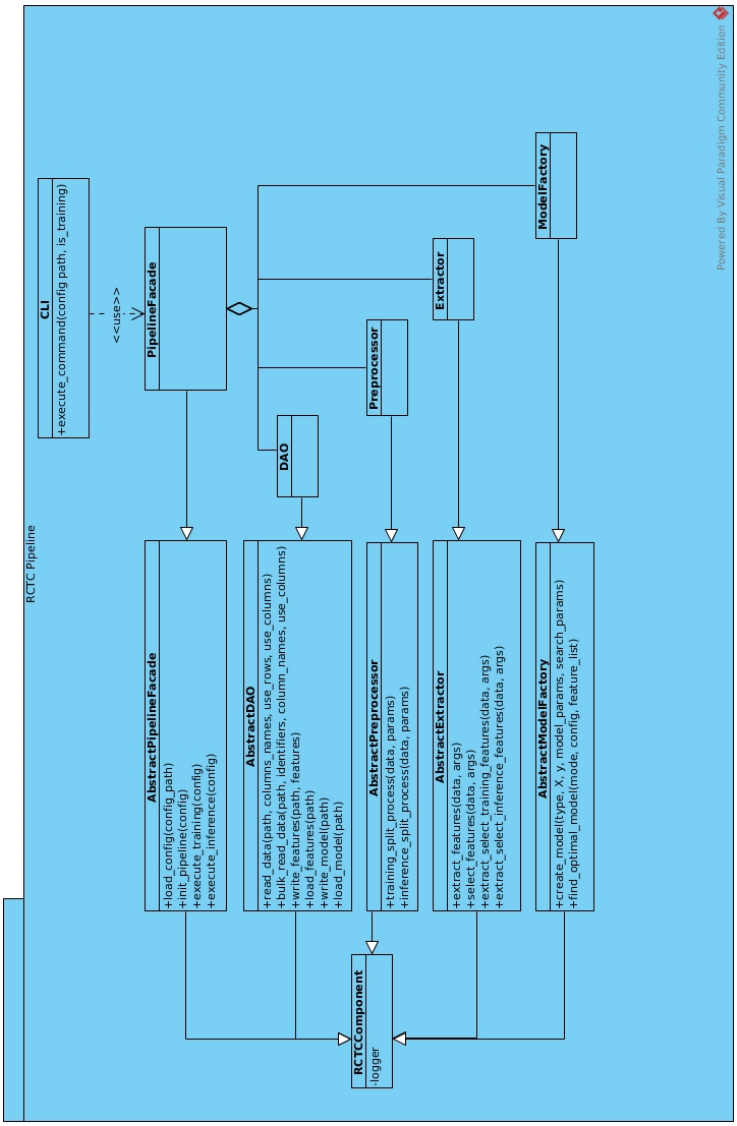


Figure 60: UML Class Diagram

C.2 Documentation

C.2.1 User Interface

A simple user interface is provided in the form of a Command Line Interface (CLI) that takes one option and one argument. The options are "–training" or "–inference" for training or inference mode and the argument "config_path" lets the software know where the config file is located. An example invocation

would be:

```
python cli.py --training config_path="~/path/to/config.json"
```

C.2.2 Config

The config file is JSON format. The following listing contains an annotated example config file for the Sussex-Huawei dataset with comments marked by # for each variable. Remove comments before use.

```
{
   "hw_num_processors": 32,          #Integer > 1
                                     #Number of
                                     #available CPUs.
                                     #Used by
                                        extraction and
                                     #classification
                                        algorithms.

  "data_set_type": "sussex_huawei", #String
                                     #Name of the
                                        dataset.
                                     #Used by the
                                        pipeline to
                                     #determine which
                                        preprocessor
                                     #to use.

  "data_set_trips": ["010317"],     #[String]
                                     #Name of the trips
                                        .
                                     #Used by the DAO
                                        to
                                     #determine which
                                        trips
                                     #to load.

  "data_set_path": "path",          #String
                                     #Path to dataset.
                                     #Used by DAO.

  "data_labels_path": "path",       #String
                                     #Path to dataset.
                                     #Used by DAO.

  "data_set_columns": [             #[Integer]
    0                                #Used by DAO to
```

```
30    ],                                    #determine which
31                                          #columns of the
32                                          #dataset to use.
33
34    "data_set_column_names": [            #[String]
35      "time",                             #Used by DAO to
36    ],                                    #determine the
          names
37                                          #of the columns
38                                          #selected by
39                                          #data_set_columns.
40
41    "data_label_columns": [               #[Integer]
42      1,                                  #Used by DAO to
43    ],                                    #determine which
44                                          #columns of the
45                                          #labels set to use
                                                .
46
47    "data_label_column_names": [          #[String]
48      "coarse_label",                     #Used by DAO
49    ],                                    #to determine the
          names
50                                          #of the columns
51                                          #selected by
52                                          #
                                                data_label_columns
                                                .
53
54    "pre_proc_validation_sz": 0.05,       #Float
55                                          #Used by
                                                preprocessor
56                                          #determine
                                                validation
57                                          #set size.
58
59    "pre_proc_training_sz": 0.95,         #Float
60                                          #Used by
                                                preprocessor
61                                          #determine
                                                training
62                                          #set size.
63
64    "pre_proc_resample_freq":             #String
65    "1000ms",                             #Used by
          preprocessor
```

```
                                    #to determine resampling
                                    #frequency.

  "pre_proc_movement_type_label":   #[Integer]
  [                                 #Specifies which activity
    5                               #labels of the dataset
  ],                                #are relevant.
                                    #Used by preprocessor
                                    #and classifiers.
                                    #For Sussex-Huawei,
                                    #5 = "Car".


  "pre_proc_road_type_label":       #[Integer]
  [                                 #Specifies which road
    1, 3                            #labels of the dataset
  ],                                #are relevant.
                                    #Used by preprocessor
                                    #and classifiers.
                                    #For Sussex-Huawei,
                                    #1,3 are "City", "Country".

  "feature_eng_extractor_type":     #String
  "motif",                          #Specifies which feature
                                    #extraction strategy
                                    #to use. Allowed
                                    #values currently
                                    #"motif" and "tsfresh".
                                    #Used by feature extractor.

  "feature_eng_mp_extractor_radii": #[Integer]
  [                                 #Specifies in
```

```
    scrimp
    10,                                 #motif discovery
    100,                                #how far two
  ],                                    #motifs must at
                                        #least be apart.
                                        #Used by motif
                                        #feature extractor
                                            .

  "feature_eng_mp_extractor_lengths": #[Integer]
  [                                     #Specifies in
    scrimp
    300,                                #motif discovery
    600                                 #how long the
  ],                                    #target motifs
                                        #should be.
                                        #Used by motif
                                        #feature extractor
                                            .

  "feature_eng_baseline_extractor_segement_len" : 60,
    #Integer
                                        #Specifies the
                                            length of a
                                                  #segment.

  "feature_eng_baseline_extractor_fdr" : 0.0001, #
    Float
                                        #Specifies TSFRESH
                                             FDR
                                        #level.

  "feature_eng_dim_reduction_type": #String
  "euclidean",                          #Specifies which
                                        #dimensionality
                                            reduction
                                        #technique is used
                                            .
                                        #Used by
                                            preprocessor.

  "feature_eng_dim_reduction_target_col_name": #String
  "acceleration_abs",                   #Name of the
    column that
                                        #contains the
                                            reduced
```

```
                                    #features. Used by
                                    #Preprocessor.

  "feature_eng_label_target_col_name": #String
  "road_label",                     #Name of the
     column
                                    #that contains
                                    #the labels after
                                    #feature
                                       extraction.
                                    #Used by extractor
                                       .

  "classifier_model_factory_type":  #String
  "sklearn",                        #Model factory
     type.
                                    #Used by model
                                       factory.
                                    #Allowed values
                                    #currently
                                    #"sklearn".

  "classifier_optimal_search_space": #[String]
  [                                 #Models to
     evaluate
    "sklearn_cart",                 #by the model
       factory.
  ],                                #Allowed values
                                    #currently "
                                       sklearn_cart",
                                    #"sklearn_rf", "
                                       sklearn_mlp",
                                    #"sklearn_svm".

  "classifier_hypermaram_space_sklearn_mlp": #JSON
                                    #Hyper param
                                       search space
                                    #for sklearn_mlp.
                                    #Used by model
                                       factory.
    {
      "test_set_sz": 0.2,           #Float
                                    #Test set size.

      "verbose": 0,                 #Integer
                                    #Verbosity. 0 =
```

```
                                        low, 10 = max.
165
166     "cross_validation_k": 10,       #Integer
167                                     #Number of splits
                                            (k) in
168                                     #k-fold
                                            crossvalidation
                                            ..
169
170     "iterations": 50,               #Integer
171                                     #Iterations over
                                            search
172                                     #space.
173
174     "save_classifier": true,        #Bool
175                                     #Store classifier
                                            on disk.
176
177     "save_classifier_file_name":    #String
178     "mlp_rs.pickle",                #Name of
           classifier on disk.
179
180     "solver": [                     #[String]
181       "adam",                       #Solvers to
             consider
182       "lbfgs",                      #during hyper
             param
183       "sgd"                         #optimization.
184     ],
185
186     "max_iter": [                   #[Integer]
187       1,                            #Range of the
             linear
188       250                           #distribution
             hyper params.
189     ],                              #[from, to].
190
191     "alpha_exponent": 10,           #Integer
192                                     #Exponent of the
                                            range
193                                     #of the linear
                                            distribution
194                                     #that alpha is
                                            drawn from.
195
196     "architectures": [              #[[Integer]]
```

```
197         [                               #Defines MLP
               architectures.
198           32,                           #Each array
                 contains
199           32                            #information about
                  the number
200         ],                              #of nodes per
               layer, first
201         [                               #value is the
               topmost
202           16,                           #layer.
203           16
204         ]
205       ],
206
207       "activation_function": [          #[String]
208         "logistic",                     #Activation
               functions
209         "relu",                         #of the MLP hyper
               param
210         "tanh"                          #search space.
211       ],
212
213       "learning_rate": [                #[String]
214         "constant",                     #Learning rates of
215         "invscaling",                   #the MLP hyper
               param
216         "adaptive"                      #search space.
217       ],
218
219       "learning_rate_init_exponent": #Exponent of the
             range
220         10,                           #of the linear
               distribution
221                                       #that learning
                                             rate is drawn
222                                       #from.
223
224       "batch_size": [                   #[Integer]
225         1,                              #Batch sizes range
               ,
226         10                            #[from, to].
227       ],
228
229       "shuffle": [                      #[Bool]
230         true,                           #Shuffle,
```

```
        false                           #True or False.
      ],

      "early_stopping": [               #[Bool]
        true,                           #Stop optimization
        false                           #before
           convergence,
      ],                                #True or False.

      "random_state": [                 #[Integer]
        1,                              #Random state
           range,
        10                              #[from, to].
      ]
    },

  "classifier_hypermaram_space_sklearn_svc": { #JSON
                                        #Hyper param
                                           search space
                                        #for sklearn_svc.
                                        #Used by model
                                           factory.

    "test_set_sz": 0.2,                 #Float
                                        #Test set size.

      "verbose": 0,                     #Integer
                                        #Verbosity. 0 =
                                           low, 10 = max.

      "cross_validation_k": 10,         #Integer
                                        #Number of splits
                                           (k) in
                                        #k-fold
                                           crossvalidation
                                           .

      "iterations": 50,                 #Integer
                                        #Iterations over
                                           search
                                        #space.

      "save_classifier": true,          #Bool
                                        #Store classifier
                                           on disk.
```

```
267       "save_classifier_file_name":  #String
268       "svc_rs.pickle",              #Name of
              classifier on disk.
269
270     "kernel": [                     #[String]
271       "rbf",                        #Kernels the SVC
272       "linear",                     #is supposed to
              use.
273       "poly"
274     ],
275
276     "degree": [                     #[Integer]
277       2,                            #Degree of
              polynomial,
278       10                            #[from, to].
279     ],
280
281     "gamma_exponent": 10,           #Integer
282                                     #Exponent of the
                                            range
283                                     #of the linear
                                            distribution
284                                     #that gamma is
                                            drawn from.
285
286     "C": [                          #[Integer]
287       2,                            #Value of C,
288       5000                          #[from, to].
289     ],
290
291     "max_iter": [                   #[Integer]
292         1,                          #Range of the
                linear
293         250                         #distribution
                hyper params.
294       ],                            #[from, to]
295
296     "shrinking": [                  #[Bool]
297       true,                         #Shrinking,
298       false                         #True or False.
299     ],
300
301     "probability": [                #[Bool]
302       true,                         #Probability,
303       false                         #True or False.
304     ],
```

```
305
306     "random_state": [                   #[Integer]
307       1,                                #Random state
              range,
308       10                                #[from, to].
309     ]
310   },
311
312   "classifier_hypermaram_space_sklearn_rf": { #JSON
313                                         #Hyper param
                                                search space
314                                         #for sklearn_svc.
315                                         #Used by model
                                                factory.
316
317     "test_set_sz": 0.2,                 #Float
318                                         #Test set size.
319
320       "verbose": 0,                     #Integer
321                                         #Verbosity. 0 =
                                                low, 10 = max
322
323       "cross_validation_k": 10,         #Integer
324                                         #Number of splits
                                                (k) in
325                                         #k-fold
                                                crossvalidation
                                                .
326
327       "iterations": 50,                 #Integer
328                                         #Iterations over
                                                search
329                                         #space.
330
331       "save_classifier": true,          #Bool
332                                         #Store classifier
                                                on disk
333
334       "save_classifier_file_name":      #String
335       "rf_rs.pickle",                   #Name of
              classifier on disk
336
337     "n_estimators": [                   #[Integer]
338       1,                                #Number of
              estimator
339       100                               #range.
```

```
    ],                                  #[from, to].

    "max_depth": [                      #[Integer]
      1,                                #Max tree depth
      128                               #range.
    ],                                  #[from, to].

    "min_samples_split": [              #[Integer]
      2,                                #Number of min
      10                                #samples range.
    ],                                  #[from, to].

    "bootstrap": [                      #[Bool]
      true,                             #Bootstrap,
      false                             #True or False.
    ],

    "criterion": [                      #[String]
      "gini",                           #Split criterion,
      "entropy"                         #"gini", "entropy
         ".
    ],

    "random_state": [                   #[Integer]
      1,                                #Random state
         range,
      10                                #[from, to].
    ]

  },
  "classifier_hypermaram_space_sklearn_cart": { #JSON
                                        #Hyper param
                                           search space
                                        #for sklearn_svc.
                                        #Used by model
                                           factory.

    "test_set_sz": 0.2,                 #Float
                                        #Test set size.

      "verbose": 0,                     #Integer
                                        #Verbosity. 0 =
                                           low, 10 = max

      "cross_validation_k": 10,         #Integer
                                        #Number of splits
```

```
                                        (k) in
381                                   #k-fold
                                        crossvalidation
                                        .
382
383       "iterations": 50,           #Integer
384                                   #Iterations over
                                        search
385                                   #space.
386
387       "save_classifier": true,    #Bool
388                                   #Store classifier
                                        on disk
389
390       "save_classifier_file_name":  #String
391       "cart_rs.pickle",             #Name of
             classifier on disk
392
393     "max_depth": [                #[Integer]
394       1,                          #Max tree depth
395       128                         #range.
396     ],                            #[from, to].
397
398     "random_state": [             #[Integer]
399       1,                          #Random state
             range,
400       10                          #[from, to].
401     ]
402
403     "criterion": [                #[String]
404       "gini",                     #Split criterion,
405       "entropy"                   #"gini", "entropy
             ".
406     ],
407
408     "splitter": [                 #[String]
409       "best",                     #Splitter,
410       "random"                    #"best", "random".
411     ],
412
413     "min_samples_split": [        #[Integer]
414       2,                          #Number of min
415       10                          #samples range.
416     ],                            #[from, to].
417   }
418 }
```

C.2.3 External Dependencies

Requirements.txt

```
attrs==19.3.0
backcall==0.1.0
bleach==3.1.0
certifi==2019.9.11
chardet==3.0.4
Click==7.0
cloudpickle==1.2.2
cycler==0.10.0
dask==2.6.0
decorator==4.4.1
defusedxml==0.6.0
dill==0.3.1.1
distributed==2.6.0
entrypoints==0.3
future==0.18.2
HeapDict==1.0.1
idna==2.8
importlib-metadata==0.23
ipykernel==5.1.3
ipython==7.9.0
ipython-genutils==0.2.0
ipywidgets==7.5.1
jedi==0.15.1
Jinja2==2.10.3
joblib==0.14.0
jsonschema==3.1.1
jupyter==1.0.0
jupyter-client==5.3.4
jupyter-console==6.0.0
jupyter-core==4.6.1
kiwisolver==1.1.0
llvmlite==0.30.0
MarkupSafe==1.1.1
matplotlib==3.1.1
matrixprofile-ts==0.0.9
mistune==0.8.4
more-itertools==7.2.0
msgpack==0.6.2
multiprocess==0.70.9
nbconvert==5.6.1
```

```
nbformat==4.4.0
notebook==6.0.2
numba==0.46.0
numpy==1.17.3
overrides==2.5
pandas==0.23.4
pandocfilters==1.4.2
parso==0.5.1
pathos==0.2.5
patsy==0.5.1
pexpect==4.7.0
pickleshare==0.7.5
pox==0.2.7
ppft==1.6.6.1
prometheus-client==0.7.1
prompt-toolkit==2.0.10
psutil==5.6.5
ptyprocess==0.6.0
Pygments==2.4.2
pyparsing==2.4.4
pyrsistent==0.15.5
python-dateutil==2.8.1
pytz==2019.3
PyYAML==5.1.2
pyzmq==18.1.0
qtconsole==4.5.5
requests==2.22.0
scikit-learn==0.21.3
scipy==1.3.1
seaborn==0.9.0
Send2Trash==1.5.0
six==1.13.0
sklearn==0.0
sortedcontainers==2.1.0
statsmodels==0.10.1
tblib==1.5.0
terminado==0.8.2
testpath==0.4.4
toolz==0.10.0
tornado==6.0.3
tqdm==4.37.0
traitlets==4.3.3
tsfresh==0.12.0
tslearn==0.2.5
urllib3==1.25.6
wcwidth==0.1.7
```

```
webencodings==0.5.1
widgetsnbextension==3.5.1
zict==1.0.0
zipp==0.6.0
```

C.2.4 Github Repo

`https://github.com/lorenz0890/road_condition_classification`

C.2.5 Hardware

Excerpt from lshw

```
description: Computer
    product: SYS-7049GP-TRT (To be filled by O.E.M.)
    vendor: Supermicro
    version: 0123456789
    serial: E291427X8901361
    width: 64 bits
    capabilities: smbios-3.1 dmi-3.1 smp vsyscall32
    configuration: boot=normal family=To be filled by O.E.M. sku=To be filled by
  *-core
       description: Motherboard
       product: X11DPG-QT
       vendor: Supermicro
       physical id: 0
       version: 1.02
       serial: VM187S021063
       slot: To be filled by O.E.M.
     *-firmware
          description: BIOS
          vendor: American Megatrends Inc.
          physical id: 0
          version: 2.1
          date: 07/20/2018
          size: 64KiB
          capacity: 15MiB
          capabilities: pci upgrade shadowing cdboot bootselect socketedrom edd
     *-memory:0
          description: System Memory
          physical id: 21
          slot: System board or motherboard
          capacity: 2304GiB
          capabilities: ecc
          configuration: errordetection=ecc
        *-bank:0
```

```
          description: DIMM DDR4 Synchronous 2666 MHz (0.4 ns)
          product: M393A2K40CB2-CTD
          vendor: Samsung
          physical id: 0
          serial: 3948CDDF
          slot: P1-DIMMA1
          size: 16GiB
          width: 64 bits
          clock: 2666MHz (0.4ns)
     *-bank:2
          description: DIMM DDR4 Synchronous 2666 MHz (0.4 ns)
          product: M393A2K40CB2-CTD
          vendor: Samsung
          physical id: 2
          serial: 3948CB8D
          slot: P1-DIMMB1
          size: 16GiB
          width: 64 bits
          clock: 2666MHz (0.4ns)
     *-bank:3
          description: DIMM DDR4 Synchronous 2666 MHz (0.4 ns)
          product: M393A2K40CB2-CTD
          vendor: Samsung
          physical id: 3
          serial: 3948D068
          slot: P1-DIMMC1
          size: 16GiB
          width: 64 bits
          clock: 2666MHz (0.4ns)
*-memory:1
     description: System Memory
     physical id: 28
     slot: System board or motherboard
     capacity: 2304GiB
     capabilities: ecc
     configuration: errordetection=ecc
     *-bank:0
          description: DIMM DDR4 Synchronous 2666 MHz (0.4 ns)
          product: M393A2K40CB2-CTD
          vendor: Samsung
          physical id: 0
          serial: 3948D455
          slot: P1-DIMMD1
          size: 16GiB
          width: 64 bits
          clock: 2666MHz (0.4ns)
```

```
   *-bank:2
        description: DIMM DDR4 Synchronous 2666 MHz (0.4 ns)
        product: M393A2K40CB2-CTD
        vendor: Samsung
        physical id: 2
        serial: 3948D6F5
        slot: P1-DIMME1
        size: 16GiB
        width: 64 bits
        clock: 2666MHz (0.4ns)
   *-bank:3
        description: DIMM DDR4 Synchronous 2666 MHz (0.4 ns)
        product: M393A2K40CB2-CTD
        vendor: Samsung
        physical id: 3
        serial: 3948CD8A
        slot: P1-DIMMF1
        size: 16GiB
        width: 64 bits
        clock: 2666MHz (0.4ns)
*-cache:0
     description: L1 cache
     physical id: 45
     slot: L1 Cache
     size: 1MiB
     capacity: 1MiB
     capabilities: synchronous internal write-back instruction
     configuration: level=1
*-cache:1
     description: L2 cache
     physical id: 46
     slot: L2 Cache
     size: 16MiB
     capacity: 16MiB
     capabilities: synchronous internal varies unified
     configuration: level=2
*-cache:2
     description: L3 cache
     physical id: 47
     slot: L3 Cache
     size: 22MiB
     capacity: 22MiB
     capabilities: synchronous internal varies unified
     configuration: level=3
*-cpu:0
     description: CPU
```

```
product: Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz
vendor: Intel Corp.
physical id: 48
bus info: cpu@0
version: Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz
slot: CPU1
size: 2800MHz
width: 64 bits
clock: 100MHz
configuration: cores=16 enabledcores=16 threads=32
```

References

[1] Agrawal, R., Faloutsos, C., and Swami, A. Efficient similarity search in sequence databases. In *International conference on foundations of data organization and algorithms* (1993), Springer, pp. 69–84.

[2] Allouch, A., Koubâa, A., Abbes, T., and Ammar, A. Roadsense: Smartphone application to estimate road conditions using accelerometer and gyroscope. *IEEE Sensors Journal 17*, 13 (2017), 4231–4238.

[3] Alluhaibi, S., Al-Din, M., and Moyaid, A. Driver behavior detection techniques: A survey. *Int. J. Appl. Eng. Res 13* (2018), 8856–8861.

[4] Aminikhanghahi, S., and Cook, D. J. A survey of methods for time series change point detection. *Knowledge and information systems 51*, 2 (2017), 339–367.

[5] Austria, S. Straßenverkehrsunfälle jahresergebnisse 2018. *Statistik Austria* (2018).

[6] Beaudoin, P., Coros, S., van de Panne, M., and Poulin, P. Motion-motif graphs. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2008), pp. 117–126.

[7] Breiman, L. Random forests. *Machine learning 45*, 1 (2001), 5–32.

[8] Chan, J. C.-W., and Paelinckx, D. Evaluation of random forest and adaboost tree-based ensemble classification and spectral band selection for ecotope mapping using airborne hyperspectral imagery. *Remote Sensing of Environment 112*, 6 (2008), 2999–3011.

[9] Chan, K.-P., and Fu, A. W.-C. Efficient time series matching by wavelets. In *Proceedings 15th International Conference on Data Engineering (Cat. No. 99CB36337)* (1999), IEEE, pp. 126–133.

[10] Christ, M., Braun, N., Neuffer, J., and Kempa-Liehr, A. W. Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing 307* (2018), 72–77.

[11] Christ, M., Kempa-Liehr, A. W., and Feindt, M. Distributed and parallel time series feature extraction for industrial big data applications. *arXiv preprint arXiv:1610.07717* (2016).

[12] Christ, et al.. TSFRESH documentation. `https://tsfresh.readthedocs.io/en/latest/text/list_of_features.html`. [Online; accessed January 08, 2020].

[13] Claudia Plant, M. P. Dm-2-hddata.featureselection. `https://moodle.univie.ac.at/pluginfile.php/8483654/mod_resource/content/1/DM-2-HDData.FeatureSelection.pdf`. [Online; accessed January 11, 2020].

[14] Cortes, C., and Vapnik, V. Support-vector networks. *Machine learning 20*, 3 (1995), 273–297.

[15] Damian Avila, e. a. Jupyter notebook. `https://jupyter.org/`. [Online; accessed January 10, 2020].

[16] Dau, H. A., and Keogh, E. Matrix profile v: A generic technique to incorporate domain knowledge into motif discovery. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2017), pp. 125–134.

[17] (Destatis), S. B. Das straßenverkehrsunfallgeschehen 2017 im überblick. *Statistisches Bundesamt (Destatis)* (2018).

[18] Gamma, E. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.

[19] Gao, Y., and Lin, J. Efficient discovery of variable-length time series motifs with large length range in million scale time series. *arXiv preprint arXiv:1802.04883* (2018).

[20] García-Gonzalo, E., Fernández-Muñiz, Z., Garcia Nieto, P. J., Sánchez, A., and Menéndez, M. Hard-rock stability analysis for span design in entry-type excavations with learning classifiers. *Materials 9* (06 2016), 531.

[21] Ge, X., and Smyth, P. Deformable markov model templates for time-series pattern matching. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining* (2000), pp. 81–90.

[22] Gers, F. A., and Schmidhuber, J. Recurrent nets that time and count. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium* (2000), vol. 3, IEEE, pp. 189–194.

[23] Gharghabi, S., Ding, Y., Yeh, C.-C. M., Kamgar, K., Ulanova, L., and Keogh, E. Matrix profile viii: Domain agnostic online semantic segmentation at superhuman performance levels. In *2017 IEEE International Conference on Data Mining (ICDM)* (2017), IEEE, pp. 117–126.

[24] Gjoreski, H., Ciliberto, M., Wang, L., Morales, F. J. O., Mekki, S., Valentin, S., and Roggen, D. The university of sussex-huawei locomotion and transportation dataset for multimodal analytics with mobile devices. *IEEE Access 6* (2018), 42592–42604.

[25] Grabocka, J., Schilling, N., Wistuba, M., and Schmidt-Thieme, L. Learning time-series shapelets. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (2014), pp. 392–401.

[26] Guido van Rossum, Barry Warsaw, N. C. Pep8. `https://www.python.org/dev/peps/pep-0008/`. [Online; accessed January 11, 2020].

[27] Handel, P., Skog, I., Wahlstrom, J., Bonawiede, F., Welch, R., Ohlsson, J., and Ohlsson, M. Insurance telematics: Opportunities and challenges with the smartphone solution. *IEEE Intelligent Transportation Systems Magazine 6*, 4 (2014), 57–70.

[28] Husnjak, S., Perakovic, D., Forenbacher, I., and Mumdziev, M. Telematics system in usage based motor insurance. *Procedia Engineering 100* (02 2015), 816–825.

[29] Janko, V., Rešçiç, N., Mlakar, M., Drobnič, V., Gams, M., Slapničar, G., Gjoreski, M., Bizjak, J., Marinko, M., and Luštrek, M. A new frontier for activity recognition: The sussex-huawei locomotion challenge. In *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers* (2018), pp. 1511–1520.

[30] Joubert, J. W., De Beer, D., and De Koker, N. Combining accelerometer data and contextual variables to evaluate the risk of driver behaviour. *Transportation research part F: traffic psychology and behaviour 41* (2016), 80–96.

[31] Keogh, et al. Matrixprofile TS. `https://github.com/target/matrixprofile-ts`. [Online; accessed January 08, 2020].

[32] Lin, J., Keogh, E., Wei, L., and Lonardi, S. Experiencing sax: a novel symbolic representation of time series. *Data Mining and knowledge discovery 15*, 2 (2007), 107–144.

[33] Linardi, M., Zhu, Y., Palpanas, T., and Keogh, E. Matrix profile x: Valmod-scalable discovery of variable-length motifs in data series. In *Proceedings of the 2018 International Conference on Management of Data* (2018), pp. 1053–1066.

[34] Liu, F. T., Ting, K. M., and Zhou, Z.-H. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining* (2008), IEEE, pp. 413–422.

[35] Lonardi, J., and Patel, P. Finding motifs in time series. In *Proc. of the 2nd Workshop on Temporal Data Mining* (2002), pp. 53–68.

[36] Lu, H., Yang, J., Liu, Z., Lane, N. D., Choudhury, T., and Campbell, A. T. The jigsaw continuous sensing engine for mobile phone applications. In *Proceedings of the 8th ACM conference on embedded networked sensor systems* (2010), pp. 71–84.

[37] Mack Sweeney. Motif-Classify. `https://github.com/macks22/motif-classify`. [Online; accessed January 10, 2020].

[38] Marius, P., Balas, V., Perescu-Popescu, L., and Mastorakis, N. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems 8* (07 2009).

[39] Martin, R. C. *Agile software development: principles, patterns, and practices*. Prentice Hall, 2002.

[40] Martin, R. C. *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009.

[41] Mendeley. Mendeley. `https://www.mendeley.com/`. [Online; accessed January 16, 2020].

[42] Microsystems, .-. S. Core j2ee patterns - data access object. `https://www.oracle.com/technetwork/java/dataaccessobject-138824.html`. [Online; accessed January 10, 2020].

[43] Mladenić, D. Feature selection for dimensionality reduction. In *International Statistical and Optimization Perspectives Workshop" Subspace, Latent Structure and Feature Selection"* (2005), Springer, pp. 84–102.

[44] Pal, S. K., and Mitra, S. Multilayer perceptron, fuzzy sets, classifiaction.

[45] 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., and Duchesnay, E. Scikit-learn: Machine learning in Python. `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`. [Online; accessed January 10, 2020].

[46] 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., and Duchesnay, E. Scikit-learn: Machine learning in Python. `https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`. [Online; accessed January 10, 2020].

[47] 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., and Duchesnay, E. Scikit-learn: Machine learning in Python. `https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC`. [Online; accessed January 10, 2020].

[48] 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., and Duchesnay, E. Scikit-learn: Machine learning in Python. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html. [Online; accessed January 24, 2020].

[49] 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., and Duchesnay, E. Scikit-learn: Machine learning in Python. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html. [Online; accessed January 11, 2020].

[50] 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., and Duchesnay, E. Scikit-learn: Machine learning in Python. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html. [Online; accessed January 14, 2020].

[51] 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., and Duchesnay, E. Scikit-learn: Machine learning in Python. https://scikit-learn.org/stable/modules/model_evaluation.html#accuracy-score. [Online; accessed January 24, 2020].

[52] 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., and Duchesnay, E. Scikit-learn: Machine learning in Python. https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html. [Online; accessed January 16, 2020].

[53] Predic, B., and Stojanovic, D. Localized processing and analysis of accelerometer data in detecting traffic events and driver behaviour. *J. UCS 18*, 9 (2012), 1152–1176.

[54] R, A. Applying random forest (classification) — machine learning algorithm from scratch with real datasets. https://medium.com/@ar.ingenious/

applying-random-forest-classification-machine-learning-algorithm-from-scratch-with-real
[Online; accessed March 10, 2020].

[55] Saeys, Y., Inza, I., and Larrañaga, P. A review of feature selection techniques in bioinformatics. *bioinformatics 23*, 19 (2007), 2507–2517.

[56] Seibt, P. *Algorithmic Information Theory*. Springer, 2006.

[57] Senin, P., Lin, J., Wang, X., Oates, T., Gandhi, S., Boedihardjo, A. P., Chen, C., and Frankenstein, S. Grammarviz 3.0: Interactive discovery of variable-length time series patterns. *ACM Transactions on Knowledge Discovery from Data (TKDD) 12*, 1 (2018), 1–28.

[58] Stavrianos. Large scale transport mode classification from mobile sensor data. https://www.sentiance.com/2017/12/04/transport-mode-classification/#Sampling_and_pre-processing, 2017. [Online; accessed January 11, 2020].

[59] Steinberg, D. Cart: classification and regression trees. In *The top ten algorithms in data mining*. Chapman and Hall/CRC, 2009, pp. 193–216.

[60] Wahlström, J., Skog, I., and Händel, P. Smartphone-based vehicle telematics: A ten-year anniversary. *IEEE Transactions on Intelligent Transportation Systems 18*, 10 (2017), 2802–2825.

[61] Ye, L., and Keogh, E. Time series shapelets: a new primitive for data mining. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining* (2009), pp. 947–956.

[62] Yeh, C.-C. M., Kavantzas, N., and Keogh, E. Matrix profile vi: Meaningful multidimensional motif discovery. In *2017 IEEE International Conference on Data Mining (ICDM)* (2017), IEEE, pp. 565–574.

[63] Yeh, C.-C. M., Zhu, Y., Ulanova, L., Begum, N., Ding, Y., Dau, H. A., Silva, D. F., Mueen, A., and Keogh, E. Matrix profile i: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In *2016 IEEE 16th international conference on data mining (ICDM)* (2016), Ieee, pp. 1317–1322.

[64] Zhu, Y., Yeh, C.-C. M., Zimmerman, Z., Kamgar, K., and Keogh, E. Matrix profile xi: Scrimp++: time series motif discovery at interactive speeds. In *2018 IEEE International Conference on Data Mining (ICDM)* (2018), IEEE, pp. 837–846.

[65] Zhu, Y., Zimmerman, Z., Senobari, N. S., Yeh, C.-C. M., Funning, G., Mueen, A., Brisk, P., and Keogh, E. Matrix profile ii: Exploiting a novel algorithm and gpus to break the one hundred million barrier for time series motifs and joins. In *2016 IEEE 16th international conference on data mining (ICDM)* (2016), IEEE, pp. 739–748.