The handle http://hdl.handle.net/1887/22055 holds various files of this Leiden University dissertation.

Author: Koch, Patrick
Title: Efficient tuning in supervised machine learning
Issue Date: 2013-10-29
Chapter 3

Feature processing

The term feature originates from pattern recognition and image processing and describes information derived from the original input attributes of a dataset. Features itself are attributes again, and serve to improve the prediction of learning algorithms. E.g., in classification the goal of feature processing is to get a better class separation before the prediction model is built. Since feature processing can improve the data quality significantly, it is an important part of ML. In Sec. 2.2 and Sec. 2.3 we have introduced different methods that are capable for feature selection and feature construction respectively.

We apply these methods for two time series applications. The first application is a time series regression problem, based on stormwater prediction. Here the goal is to predict fill levels of a stormwater overflow tank in Germany only based on past rainfall data. In the past, good working features have been given by experts. Nevertheless it is unknown, if the expert features and their parametrizations can even be improved using parameter tuning and feature selection.

The second application is about classifying gestures obtained from a Nintendo Wii console. We analyze a technique from the neuroscience, the Slow Feature Analysis, for creating an improved feature set, making it possible to perform classifications solely with a simple Gaussian classifier.

We highlight the advantages of automatic and semi-automatic methods for feature construction and selection. While feature selection and construction is often applied in a separate process relaxed from the main modeling, we recommend to attach it to the learning process. It can easily be seen that with very simple techniques we can achieve better results than experts, without requiring human interactions during the learning process. We demonstrate that the combination of feature processing together with an integrated tuning can generate better feature sets, which can be seen as a first step towards a comprehensive machine learning framework, see Konen et al. [154].
3.1 Generic feature processing for time series analysis

In this section we highlight the importance of feature processing, by combining a learning algorithm with pre-processing operators. We conduct a systematic tuning for the parameters of the pre-processing methods and the learning algorithm simultaneously and show the influence of the parameter tuning on the prediction quality. The simultaneous tuning of pre-processing parameters and the parameters of the learning algorithm are usually not done in practice. However, we think that this can be an important ingredient for building powerful learning algorithms. For this reason we show how the parametrization of the pre-processing operators affects the prediction models.

3.1.1 Research questions

We define the following research questions:

Q1 Is it possible to move away from domain-specific models without loss in accuracy by applying modern machine learning methods on data augmented through generic time-series pre-processing operators?

Q2 Can tuning heuristics give insights into the strengths and weaknesses of the features generated by generic pre-processing operators?

3.1.2 Predictive control

Intelligent predictive control systems provide solutions to environmental engineering processes. In water resource management, efficient controllers of stormwater tanks prevent flooding of sewerage systems, which reduces environmental pollution. With accurate predictions of stormwater tank fill levels based on past rainfall, such controlling systems are able to detect state changes as early as possible. Up to now, good results could only be achieved by applying special-purpose models especially designed for stormwater prediction. The question arises whether it is possible to replace such specialized models with state-of-the-art machine learning methods, such as Support Vector Regression (SVR) in combination with generic time series pre-processing operators, to achieve competitive performance. This study shows that even superior results can be obtained if both SVR and pre-processing parameters are tuned, to improve the performance of the learning algorithm.

Environmental engineering offers important concepts to preserve clean water and to protect the environment. Stormwater tanks are installed to stabilize the load on the sewage system by preventing rainwater from flooding the sewage systems and by supplying a base load in dry periods. Mostly, heavy rainfalls are the reason for overflows of stormwater tanks, causing environmental pollution from wastewater contaminating the environment. To avoid such situations, the effluent (the outflow) of the stormwater tanks must be controlled effectively and possible future state changes in the inflow should be recognized as early as
The time series regression problem of predicting a stormwater tank fill-level at time \( t \) from a fixed window of past rainfall data from time \( t \) back to time \( t - W \) will be referred to as the *stormwater problem* in the remainder of this thesis.

A model that predicts fill levels by means of rainfall data can be an important aid for the controlling system. Special sensors (Fig. 3.1) record time series data which can be used to train such a model.

![Figure 3.1: Left: Rain gauge (pluviometer). Right: Stormwater tank.](image)

Again, we are aware of the large number of methods for time series analysis [31], ranging from classical statistical regression to computational statistics, but as we indicated earlier, such methods usually are limited to predict the output by means of autoregression and are only suited for univariate time series [31]. As an alternative very often special-purpose models are incorporated for solving such tasks, which are specially adopted to the problems at hand. This situation is of course very unsatisfying for the practitioner in environmental engineering, because new models have to be created and parameters have to be set again for each problem.

Therefore, it would be an advantage to use methods which are general enough to be re-applied and can easily be adapted to new problems. In this experiment we use Support Vector Machines (SVMs) [63], more precisely Support Vector Regression, as a state-of-the-art method from ML and apply them to the stormwater problem. We commemorate, that learning algorithms usually suited for classification and regression, must be adapted for the time series data first. Therefore we investigate generic pre-processing operators to embed time series data and to generate new input features for the SVM model. In addition, we use sequential parameter optimization (SPO) [8] and a Genetic Algorithm (GA) to find good hyperparameter settings for both pre-processing and SVM parameters. We analyze the robustness of our method against overfitting and oversearching [198] of hyperparameters.

Preliminary work in stormwater prediction has been done by Konen et al.[156], Bartz-Beielstein et al. [14] and Flasch et al. [79]. A conclusion of these previous publications was that good models can be obtained by using specialized models which are adapted to the
Table 3.1: Real-world time series data from a stormwater tank in Germany. Four scenarios are defined, which are consecutive measurements from April to July 2005.

<table>
<thead>
<tr>
<th>Set</th>
<th>Start Date</th>
<th>End Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set 3</td>
<td>2006-06-19 03:01:00</td>
<td>2006-07-06 11:41:00</td>
</tr>
<tr>
<td>Set 4</td>
<td>2006-07-23 20:21:00</td>
<td>2006-08-10 05:01:00</td>
</tr>
</tbody>
</table>

stormwater problem.

3.1.3 Stormwater data

Time series data was collected from a real stormwater tank in Germany and consists of 30,000 data records, ranging from April to August 2006. Rainfall data are measured in three-minute intervals by a pluviometer as shown in Fig. 3.1. As training set we always used a 5,000 record time window (Set 2, Fig. 3.2) in order to predict another 5,000 record time window for testing (Set 4) which is not directly successive with regard to time to the training period (see Tab. 3.1). Later, we conducted a hyperparameter tuning with SPOT and feature selection where we used all 4 different datasets (Set1, Set2, Set3, Set4), each containing 5,000 records to analyze the robustness of our approach against oversearching.

Figure 3.2: Time window used for training (Set 2) showing the rainfall and fill levels of the stormwater tank. It can be seen from the plot, that the rainfall depicts a very spiky curve, causing the difficult predictability of the fill levels.
3.1.4 Benchmarking time series

The prediction error on the datasets is taken as objective function for the optimization procedure and for the feature selection wrapper by means of GA. For comparing models, we calculate the root mean squared error (RMSE) as a quality measure:

\[
RMSE = \sqrt{\text{mean}(Y_{\text{predicted}} - Y_{\text{true}})^2}
\]  

(3.1)

We also incorporate the Theil’s \(U\) index of inequality [239], where the RMSE of the trained model is compared to the RMSE of a naive predictor. The advantage of Theil’s \(U\) is, that it can be better interpreted by users, whereas the RMSE is just a number depending on the problem structure and range of the output variable. For Theil’s \(U\), we are using the mean of the training data target as a naive predictor:

\[
U = \frac{\text{RMSE}(\text{model})}{\text{RMSE}(\text{naive})}
\]  

(3.2)

A value of \(U\) greater than 1 indicate models that perform worse than the naive predictor, while values smaller than 1 indicate models that perform better than the naive predictor.

3.1.5 The INT2 model

In previous works [14, 156], the stormwater tank problem was investigated with different modeling approaches, among them FIR, NARX, ESN, a dynamical system based on ordinary differential equations (ODE) and a dynamical system based on integral equations (INT2). All models in these former works were systematically optimized using SPO [8]. Among these models the INT2 approach turned out to be the best one [14]. The INT2 model is an analytical regression model based on integral equations. Disadvantages of the INT2 model are that it is a special-purpose model only designed for stormwater prediction and that it is practically expensive to obtain an optimal parameter configuration: The parameterization example presented in [14] contains 9 tunable parameters which must be set.

### Table 3.3: RMSE values of the INT2 model using hand-tuned and SPO-optimized parameter configurations.

<table>
<thead>
<tr>
<th>Testset</th>
<th>Hand-tuned</th>
<th>SPO-best</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Set</td>
<td>24.27</td>
<td>23.79</td>
</tr>
<tr>
<td>Test Set</td>
<td>10.61</td>
<td>9.00</td>
</tr>
</tbody>
</table>

---
We selected two scenarios from the stormwater dataset, each containing 5000 records. The two scenarios served as training and test data respectively (Tab. 3.2). Results obtained by INT2 on the stormwater data is presented in Tab. 3.3 for the training and test sets. We compare hand-tuned INT2 parameters with the best parameter configuration determined by SPO in an earlier study [14].

### 3.1.6 SVR and time series data

Time series can usually not be handled directly by SVR. SVR does not know the time dependencies. Therefore, we transform the time series regression problem to a regression problem and apply SVR to it. Pre-processing operators are used to add new features based on the input data. There are two possibilities to integrate such operators into the SVR learning process:

- Integration into the SVR kernel function, i.e. replacing standard kernel functions by kernel functions that incorporate pre-processing
- Direct pre-processing of the data, i.e. by augmenting the input feature set with results of pre-processing

In this work we choose the second approach, because the effect of this integration into a model is easier to analyze. In a first step, we compare the effects of applying different types of time series pre-processing operators on SVM model accuracy. Details on pre-processing operators as the embedding operator and leaky rain as an integral operator suited for time-series analysis are given in Sec. 3.1.6.2 and Sec. 3.1.6.3 respectively. All additional input features generated through these pre-processing operators were used in different variants in the following experimental setups to make them comparable:

- E1 Predicting fill levels based only on current rainfall
- E2 Predicting fill levels with embedding of rainfall
- E3 Predicting fill levels with embedding of leaky rain and rainfall
- E4 Predicting fill levels with embedding of leaky rain only
- E5 Predicting fill levels with embedding of multiple leaky rain kernel functions

In our experiments we used the radial basis SVM kernel from the e1071 SVM-implementation in R, since we achieved best results with this kernel choice. Other SVR hyperparameters were obtained by SPO, namely parameters $\gamma$ and $\epsilon$, to make our results comparable to each other. All models in our study were trained on the training time window and evaluated on the test set (see Tab. 3.2). As an objective function for SPO tuning we
used the prediction error on the test set, which is topic of criticism and will be treated later in this thesis.

In the following subsections, we describe the setup of the pre-processing operators used for these different model variants in more detail.

3.1.6.1 E1: predicting fill levels without pre-processing

The most naive approach is to predict fill levels solely based on the current rainfall, taken as only input feature for the SVM. No matter what SVR hyperparameter configuration was used, the obtained RMSE for this model on the test set could not get about 45.8% better than a naive prediction.\footnote{Naive prediction means predicting the mean value of the training set.} Although the SVM prediction is more accurate than the naive one, it can be seen in Fig. 3.3 that the length of high fill level periods are frequently underestimated throughout the whole test period. For this reason, this very simple model is not competitive to models like the INT2 by Konen et al. [156].

![Figure 3.3: Plot of predicted fill levels using only rain data without any pre-processing.](image)

\[ \Omega(r, t, W_{rain}) := (r(t), r(t - 1), \ldots, r(t - W_{rain})) \quad (3.3) \]

3.1.6.2 E2: embedding of rainfall

One major difference of time series problems in comparison to standard regression problems is that they usually have certain dependencies of successive records. This has not been taken into consideration in our last model, which might be a main reason for its poor accuracy. Of course the model should incorporate that \( l(t) \) (the fill level at time \( t \)) is somehow related to previous rainfall values \( r(t - 1), r(t - 2), \ldots, r(t - W_{rain}) \), for a positive \( W_{rain} \) (under assumption that the value \( r(t - W_{rain}) \) is available in the data). More formally an embedding \( \Omega \) of a feature \( r \) from time \( t \) with dimension \( W_{rain} \) is defined as follows:
Table 3.4: Best parameter configuration for rainfall embedding. The region of interest (RoI) bounds have been refined after preliminary runs.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>RoI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embedding $W_{rain}$</td>
<td>43</td>
<td>[5, 50]</td>
</tr>
<tr>
<td>SVM $\gamma$</td>
<td>0.012</td>
<td>[0.01, 0.03]</td>
</tr>
<tr>
<td>SVM $\epsilon$</td>
<td>0.012</td>
<td>[0.01, 0.02]</td>
</tr>
</tbody>
</table>

There should be a certain relationship between the value to predict and rainfall values (features) from the past. We analyze the extension of the input feature space $\mathcal{F}$ with its historical part $\Omega(r, t, W_{rain})$.

The influence of past data points on the current data point is generally unknown, but might be detected by the SVM model, if we augment each record $r(t)$ with its predecessors $r(t-1), \ldots, r(t-W_{rain})$. Since the embedding parameter might be important for the model quality, we add this parameter to the tuning. This setup led us to a tuning of either the embedding dimension $W_{rain}$ or of the SVM parameters $\gamma$ and $\epsilon$. The tuning algorithm SPO returned near-optimal parameter values which are presented in Tab. 3.4.

With an RMSE of 14.98 (cf. Tab. 3.7), the SVM model has gained accuracy by using an embedding of past rainfall in comparison to just using the current rainfall data point.

3.1.6.3 E3: leaky rain and rainfall embedding

In the design of the previous model, the rainfall at time $t-40$ has been considered to be of the same importance as the rainfall at time $t-2$. This is of course not true, because loosely speaking, the rainfall from two days in the past should not have the same impact on the fill level as the rainfall from the last 20 minutes. Or more precisely, the rain is a measurable quantity which drains into the soil and then — depending on the consistence of the soil — flows into the stormwater tanks with a certain delay. Therefore the rainfall could be summed up and this integrated quantity could then be used as a new feature of the input data. How can it be expressed that, given $w_2 > w_1$, rainfall from time $t - w_2$ has less influence than $t - w_1$ on $l(t)$? We define the pre-processing operator leaky rain

$$\sum_{i=0}^{T} \lambda^i \cdot r(t - i)$$

where $\lambda \in [0, 1]$, $T$ is the length of the integration and $t$ is the current time. The formula can be efficiently computed by Fast Fourier Transformation (FFT) even for large datasets.

3.1.6.4 E4: embedding of leaky rain only

Surprisingly the rainfall embedding is no longer advantageous when leaky rain embedding is taken into account. This fact can be clearly deduced from Fig. 3.4, where the RMSE is nearly monotonously increasing with higher rainfall embedding dimensions, given a constant
leaky rain embedding dimension of $W = 43$. Besides that, the modeling process is slower when higher embedding dimensions are used, because the SVM has to cope with more input dimensions and much more data.

Learning from this observation, we omit the rainfall embedding $W_{\text{rain}}$ and concentrate on optimizing the embedding dimension for the leaky rain embedding. Therefore, we performed a similar embedding dimension experiment for leaky rain. Results are shown in Fig. 3.5. The plot shows clearly that there is an almost monotonous decrease of prediction error when using embeddings up to 40 dimensions. The prediction error increases again for more than 40 dimensions. This effect could be explained by two processes. First, the information gain decreases with increasing embedding dimension, because the influence of rainfall on the target decreases with increasing time lag. This may lead to a model with worse generalization capabilities. Secondly, the model is fitted to a higher number of input dimensions, including disturbing factors as noise, measurement errors, etc. Summarized, these factors may lead to more complex models which are prone to overfitting.

Using SPO for tuning the parameters depicted in Tab 3.5 with given region of interest (RoI), SPO delivered an embedding dimension of 43 which is not the global optimum, but is very close to it. Possible improvements on this result can be achieved by increasing the function evaluations in the SPO settings or incorporating a local search strategy on the SPO parameters for fine-tuning.

**3.1.6.5 E5: two leaky rain functions**

Leaky rain has shown to be an adequate pre-processing function to simplify learning of the target function. However, the true function might be more complex due to factors not incorporated in our simple leaky rain function. Therefore, we employed a more complex pre-processing by using two leaky rain functions at the same time (Fig. 3.6). We tuned the
Figure 3.5: Progressively increasing the leaky rain embedding dimension $W$. The optimal embedding dimension is at $W = 40$ which could be well approximated by SPO. No rainfall embedding was used here.

**Table 3.5:** SPO parameter for SVM and leaky rain pre-processing

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>RoI</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma$</td>
<td>SVM radial basis kernel parameter</td>
<td>[0.005, 0.01]</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>parameter for the insensitive loss function</td>
<td>[0.005, 0.01]</td>
</tr>
<tr>
<td>$T$</td>
<td>window size for the calculation of the leaky rain sum</td>
<td>[50, 120]</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>leaky rain kernel parameter (cf. Equation 3.4)</td>
<td>[0.00001, 0.3]</td>
</tr>
<tr>
<td>$W$</td>
<td>embedding dimension for leaky rain</td>
<td>[5, 50]</td>
</tr>
</tbody>
</table>

parameters $\lambda$, $T$, and $W$ independently for each of the two functions by SPO leading to a new parameter configuration as shown in Tab. 3.6.

After integrating the two kernel functions in our model, compared to a single leaky rain functions, the RMSE slightly improves from 8.09 to 7.80, clearly outperforming the INT2 model again. Note that one leaky kernel function uses a larger $\lambda$ value and a shorter embedding dimension (less than half the size of the first embedding dimension), so that both functions seem to supplement each other (Fig. 3.6).

In this analysis we used Support Vector Machines (SVMs) as a regression tool for predicting fill levels of stormwater overflow tanks. The results of generic pre-processing are summarized in Tab. 3.7. We can give a positive answer for research question Q1. A SVM model is remarkably better than the special-purpose model INT2 under the assumption that i.) pre-processing of the data and ii.) tuning of the SVM and the pre-processing parameters is incorporated. By the tuning we could also determine how important certain features are. E.g., for the stormwater problem the embedding of rainfall did not lead to a better performing prediction model. It seems that rainfall itself does not contain enough accessible information to do accurate predictions of the fill levels. The summarized results in Tab. 3.7
3.2 Learning slow features

Feature processing has a long tradition in ML, data mining and statistical learning. Especially in tasks like time series analysis feature processing plays a key role. Here it is often unclear, how features can be derived from the data. Additionally, time series data are different compared with normal regression and classification tasks. As a consequence it is difficult to apply standard feature processing methods from ML, since they are usually proposed

Figure 3.6: Red curves: The two leaky rain kernel functions obtained by SPO. Blue lines: Zero level.

Table 3.6: Best SPO parameter configuration for all tuned parameters evaluated on the test set. The region of interest (RoI) bounds are the values found after preliminary runs.

<table>
<thead>
<tr>
<th>Parameter Type</th>
<th>Parameter</th>
<th>Best Value found</th>
<th>RoI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embedding</td>
<td>$W_1$</td>
<td>39</td>
<td>[5, 50]</td>
</tr>
<tr>
<td></td>
<td>$W_2$</td>
<td>16</td>
<td>[5, 50]</td>
</tr>
<tr>
<td></td>
<td>$T_1$</td>
<td>114</td>
<td>[50, 120]</td>
</tr>
<tr>
<td></td>
<td>$T_2$</td>
<td>102</td>
<td>[50, 120]</td>
</tr>
<tr>
<td></td>
<td>$\lambda$</td>
<td>0.0250092</td>
<td>[0.00001, 0.3]</td>
</tr>
<tr>
<td></td>
<td>$\lambda_2$</td>
<td>0.225002</td>
<td>[0.00001, 0.3]</td>
</tr>
<tr>
<td>SVM</td>
<td>$\gamma$</td>
<td>0.0116667</td>
<td>[0.005, 0.01]</td>
</tr>
<tr>
<td></td>
<td>$\epsilon$</td>
<td>0.0116667</td>
<td>[0.005, 0.01]</td>
</tr>
</tbody>
</table>

show that there is a large reduction of prediction error by using the more sophisticated leaky rain input feature: The RMSE with leaky rain embedding decreases from 14.98 to 10.14. It is an interesting side effect, that we can find out by tuning experiments, which parameters are helpful, confirming research question Q2.

As a point of discussion, it has to be criticized that our model parameters were tuned using the test set for evaluation by the optimizing algorithm (SPO). Actually, the prediction error on the test set should only be used for evaluating the model and not for a parameter tuning. Later in this thesis, we analyze if our model is robust to oversearching by introducing an unbiased estimator for time series analysis.
for classification or regression tasks and do not respect the time dependencies. Thus it is necessary to include special pre-processing — or as another option — to incorporate special-purpose models. However, in ML the majority of research is performed for standard classification and regression. But we think, that time series will become moreover important in the forthcoming years, as most industrial processes involve time as a measurement. Algorithms for modeling time series data include amongst others time series models as ARMA, or GARCH. Unfortunately these methods were only proposed for one univariate time series, but not for modeling multivariate time series. In general, all learning algorithms like Random forest or SVM are able to model multivariate time series. But as a precondition, these methods need to incorporate feature processing, which can be time-consuming and are not necessarily off-the-shelf working well. A method which is able to perform feature processing and time series modeling in one process is the SFA algorithm, which was introduced in Sec. 2.3.3. In a first study, we want to test how good it performs for a real-world classification task.

![Figure 3.7: Prediction by SVM with two leaky rain embeddings optimized by SPO.](image)

**Table 3.7:** Precision comparison of all models on testset

<table>
<thead>
<tr>
<th>Model type</th>
<th>RMSE</th>
<th>Theil’s $U$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve prediction</td>
<td>33.34</td>
<td>1.00</td>
</tr>
<tr>
<td>E1: Only Rainfall</td>
<td>18.07</td>
<td>0.54</td>
</tr>
<tr>
<td>E2: Rainfall Emb. w/o Leaky</td>
<td>14.98</td>
<td>0.45</td>
</tr>
<tr>
<td>E3: Rainfall and Leaky Rain</td>
<td>10.14</td>
<td>0.30</td>
</tr>
<tr>
<td>INT2</td>
<td>9.00</td>
<td>0.27</td>
</tr>
<tr>
<td>E4: Leaky Rain Embedding</td>
<td>8.09</td>
<td>0.24</td>
</tr>
<tr>
<td>E5: Two Leaky Embeddings</td>
<td>7.80</td>
<td>0.23</td>
</tr>
</tbody>
</table>
3.2.1 Research questions

We use SFA to classify gestures obtained from a Nintendo Wii console. The question of interest is, if SFA can be helpful in generating features for the gesture signals:

Q1 Is it possible to construct features with SFA, that a simple Gaussian classifier is able to achieve results comparable to other state-of-the-art learning algorithms, but in shorter computation time?

Q2 How good is the generalization quality of the learned features, measured by the classification accuracy of gestures of unseen persons?

The Slow Feature Analysis has been introduced in Sec. 2.3.3. We give detailed information about the acquisition and preparation of the data in Sec. 3.2.2. The experimental results are described in Sec. 3.2.4 and we give a brief conclusion in Sec. 3.2.5.

3.2.2 Introduction to gesture recognition

The integration of accelerometers in electronic devices such as mobile phones or game controllers for gesture recognition has become popular in consumer electronics. While in the 1990s, the recognition of gestures required a high demand on hardware and was only present in the labs of research institutes, gesture recognition has now made its way into the homes. Even though there already exists a large number of games for the Nintendo Wii game console, recognition of complex gestures still remains challenging. In spite of the large number of classification approaches using the Wiimote controller with the infrared device, see, e.g., the work of Lee [162], we focus here on approaches where no infrared device is used. Although thereby the task becomes more difficult, we are not so dependent on the Wii structure and can easily transfer our approach to other devices. I.e., this can be important, when infrared cannot be used for some reason and only acceleration based sensors are available. Related work has been proposed by Schlömer et al. [213], who present a classification tool based on a Hidden Markov Model [200] and Liu et al. [168] who applied personalized gesture recognition to a user authentication problem. Rehm et al. [204] describe classification of Wii gestures with different methods and expose the influence of cultural behaviours in gestures. Malmestig and Sundberg [171] use the Wiimote to recognize gestures for sign language recognition with good results.

3.2.3 Gesture recognition using accelerometer data

Nintendo invented the Wiimote Controller (Fig. 3.8) as a Bluetooth compliant game controller for the Nintendo Wii console. In its basic version an acceleration sensor, the accelerometer, is implemented inside the device. An infrared device for locating the position of the controller during games is also integrated in the Wiimote. The Wii Motion Plus device is an additional device for the Wiimote, based on two gyrometers for calculation
of angular velocities. In order to keep the analysis as simple as possible with respect to the sensor and hardware side, we neither use the sensor information from the Wii infrared sensor (sensor bar), nor from the Wii Motion Plus sensors. As observed in other studies by Schlömer et al. [213] and Rehm et al. [204] we expect to get comparably good and accurate recognition rates with the sensor data solely taken from the accelerometer.

Figure 3.8: Nintendo Wiimote Controller. The controller is designed as game controller for the Nintendo Wii game console. It contains several buttons, an infrared sensor and an accelerometer. The device is wireless, the signals are transferred via Bluetooth.

3.2.3.1 Data recording and preparation

The dataset consists of five different gestures recorded from ten test persons with the Avetana Bluetooth library\(^2\) together with a modified implementation of the Wiigee framework by Poppinga [195]. Although the number of test persons is rather small, the data shows both varieties and similarities. The acceleration values for several patterns of the frisbee gesture are exemplarily plotted in Fig. 3.9. The accelerometer sensor data were recorded with their timestamps at a rate of approximately 100 Hz. The detection rate was not always equal to 100 Hz, because the data cannot be equidistantly delivered from the Bluetooth interface. In a preliminary data preparation step we interpolated the accelerometer data at 300 equidistant time points between the first and the last timestamp. A first view on the data revealed that the patterns usually vary strongly in execution time and amplitude; even execution times from the same person seem to be rather volatile. In Tab. 3.8 we present a summary of the execution times of a single gesture set, resulting from the ten test persons. Due to the large variance we conduct a smoothing of the gestures by taking the mean of each 10 consecutive data points, resulting in a set of \( n = 30 \) points for each accelerometer dimension \( x_{\text{acc}}(t) \),

\(^2\)http://sourceforge.net/projects/avetanabt/
3.2. Learning slow features

Figure 3.9: Sensor data for all gesture patterns from a frisbee throw obtained from one person. **Left column:** \(x_{\text{acc}}, y_{\text{acc}}\) and \(z_{\text{acc}}\) acceleration sensor values before time- and amplitude-normalization. **Middle column:** Sensor values after time normalization. **Right column:** Final curves for classification after amplitude normalization.

Table 3.8: Absolute execution times (in seconds) for five example gestures performed by ten test persons. Each gesture was recorded ten times.

<table>
<thead>
<tr>
<th>Gesture</th>
<th>Minimum</th>
<th>Median</th>
<th>Maximum</th>
<th>Std.Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circle</td>
<td>0.773</td>
<td>1.607</td>
<td>4.914</td>
<td>0.343</td>
</tr>
<tr>
<td>Throw</td>
<td>0.416</td>
<td>0.957</td>
<td>2.306</td>
<td>0.471</td>
</tr>
<tr>
<td>Frisbee</td>
<td>0.376</td>
<td>0.883</td>
<td>2.036</td>
<td>0.353</td>
</tr>
<tr>
<td>Bowling</td>
<td>0.530</td>
<td>1.325</td>
<td>2.303</td>
<td>0.377</td>
</tr>
<tr>
<td>‘Z’</td>
<td>0.980</td>
<td>1.607</td>
<td>4.280</td>
<td>0.453</td>
</tr>
</tbody>
</table>

\(y_{\text{acc}}(t)\) and \(z_{\text{acc}}(t)\). Another preliminary observation was that the gestures from different persons usually vary in the size of the amplitude. Hence, as an additional operator we use an amplitude normalization by dividing the accelerometer data of each gesture by its standard deviation. In Fig. 3.9 the difference between the gesture signals before and after data preparation can be seen. It is clearly visible that despite the normalization steps were performed, there is still considerable variation within the same gesture type from the same person. Although there is only a small difference between the time-normalized gestures in the middle column and the gestures with amplitude normalization in the right column, the results became slightly better with amplitude normalization for all classifiers.

In order to produce one vector for each gesture we concatenate the multivariate time series data from the three acceleration sensors \(x_{\text{acc}}(t), y_{\text{acc}}(t), z_{\text{acc}}(t)\) into a single pattern \(\vec{X}(t) = (x_{\text{acc}}(t) \oplus y_{\text{acc}}(t) \oplus z_{\text{acc}}(t))\), where \(\oplus\) denotes concatenation, \(\vec{X}(t)\) has the dimension
$3n = 90$. We concatenate the raw gesture execution time in seconds as the last dimension, making $\vec{X}(t)$ finally a $3n + 1 = 91$-dimensional vector.

Since 91 dimensions as input to SFA lead to very large processing times and memory requirements, we reduced the dimensions as a final data preparation step by PCA to the $n_{pp}$ dimensions which carry most of the variance in the training data. Usually we let $n_{pp}$ vary between 3 and 20, since higher values are computationally expensive and lower values have a negative impact on the classification. In the case of RF we do not need this dimension reduction.

### 3.2.4 Experimental analysis

In this section we perform a classification of the recorded gesture signals using a random forest classifier to analyze the performance of the SFA feature construction.

#### 3.2.4.1 Experimental setup

The gestures recorded for recognition were performed ten times by each person. The participants had to push and hold a button on the Wiimote while performing the gestures. The recorded gestures were passed through the data preparation steps as described in Sec. 3.2.2. These pre-processed patterns were passed into the SFA and RF classifiers. All SFA calculations in this thesis were performed with the extended Matlab toolkit sfa-tk V2.0 from [153, 16]. For RF we used the `randomForest` R-package.³

The gesture set used in our work is thought to be almost realistic for the application in games, since a set of five common gestures is used:

- Circle,
- Throw,
- Bowling,
- Frisbee,
- Z (the letter 'Z' painted in the air)

Both SFA and RF need training patterns as input. Because the classification highly depends on the chosen division of training and test data we differ in the following between several recognition tasks with respect to test and training sets:

(A) **Random sampling:** Classification on the recorded gestures with random sampled test and training set. A 10-fold cross-validation is used to verify the results.

³[http://cran.r-project.org/web/packages/randomForest/index.html](http://cran.r-project.org/web/packages/randomForest/index.html)
3.2. Learning slow features

Figure 3.10: Error rates achieved with SFA for different pre-processed dimensions \( n_{pp} \) used after PCA preprocessing. Shown are the averages out of 10 runs with different seeds for the CV fold generation. We show for comparison the mean RF error rate 2.09\% from Tab. 3.9 (independent of \( n_{pp} \)).

(B) Recognition of unseen persons: Splitting of training and test set by leaving all gestures from a certain person out of the training set (holdout set). The partitioning is done for all persons sequentially.

3.2.4.2 Random sampling

In figure 3.10 and table 3.9 we show results of a 10-fold cross-validation (CV) from ten independent runs on randomly sampled training data. The 716 gestures were divided into 10 folds containing 71 or 72 records. Each fold in turn was considered as test data while the remaining data were training data. The test error rate on unseen gestures converges quickly to values around 2\% as the PCA-reduced dimension attains \( n_{pp} = 12 \) or above. To measure the impact of the SFA features we directly trained a Gaussian classifier on the same pre-processed input of dimension \( n_{pp} \) and tested its performance on the same unseen test data. For \( n_{pp} \geq 12 \) the pure Gaussian classifier is worse by a factor of 6, showing the strength of the feature combinations found by SFA. We also show in Fig. 3.10 and Tab. 3.9 the results of RF (blue dash-dotted line). RF is slightly better for \( n_{pp} = 12 \), but for \( n_{pp} = 15 \) and above SFA and RF are very similar.
Table 3.9: Error rates obtained from ten runs of 10-fold cross-validation on random sampled gesture data. Values in bold are best.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Best</th>
<th>Mean</th>
<th>Worst</th>
<th>Std.Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFA($n_{pp}$=12)</td>
<td>2.37</td>
<td>2.82</td>
<td>3.21</td>
<td>0.25</td>
</tr>
<tr>
<td>SFA($n_{pp}$=15)</td>
<td>1.68</td>
<td>2.03</td>
<td>2.24</td>
<td>0.18</td>
</tr>
<tr>
<td>RF</td>
<td>1.54</td>
<td>2.09</td>
<td>2.37</td>
<td>0.30</td>
</tr>
<tr>
<td>Gauss</td>
<td>13.55</td>
<td>14.02</td>
<td>14.39</td>
<td>0.22</td>
</tr>
</tbody>
</table>

3.2.4.3 Classifying gestures from unseen persons

In classification with random sampling of training and test sets it is indirectly assumed that the training data is representative for the test set. However, in gesture recognition it is of high importance that gestures of persons who never occurred in the training set can also be recognized by the classifier. Rehm et al. [204] highlighted that gestures can be influenced by the expressivity of the user. Factors for expressivity are, e.g., speed, space used for the gesture and the cultural background. Due to the large differences between several persons this might be a quite difficult task for any classifier. Nevertheless this experiment can be of high relevance for the game industry, no person-specific calibration will be needed in case of a good classification.

Figure 3.11: Error rates when classifying gestures of persons unseen in the training set (SFA and parametric bootstrap (PB), Gauss: Gaussian Classifier, RF: Random Forest).

In Fig. 3.11 we show the results of a cross-validation experiment for SFA with parametric bootstrap (SFA+PB), a Gaussian classifier and RF when the gestures of each person in
3.2. Learning slow features

![Graph](image)

(a) SFA error rates

(b) RF error rates

Figure 3.12: (a) Error rates achieved with SFA in the interval of [40, 200] training patterns (parameter setting: $n_{pp} = 12$). The test set error suddenly increases when the number of patterns is too small for a sufficient rank of the covariance matrix (< 90 patterns), while the error on the training set stays constantly near zero. (b) The RF error rates also increase with fewer training patterns, but do not rise to an unnatural high level in contrast to SFA. Additionally, the training set error is a good predictor for the test set error on unseen data.

turn are put completely into the test set. As expected, the error rates are considerably higher than with random sampling. Tab. 3.10 shows the error rate when averaging over all gestures which are 15.3% for SFA, 14.8% for RF, and 19.6% for Gauss. But the most striking attribute of Fig. 3.11 is the great variety of error rates between persons. The gestures of some persons from our dataset could be extremely well classified for both SFA and RF, while others, most prominently gestures of the person with ID 128, were hard to classify for all algorithms. Presumably this person has very different characteristics when performing gestures, but further research is necessary to get more insights for this.

3.2.5 Conclusion

We present the best error rates in Tab. 3.10 for the tasks (A) and (B) from Sec. 3.2.4 for our three algorithms. For SFA we chose settings $n_{pp} = 12$, $\sigma_{nc} = 0.8$ plus $N_{copies} \in \{0, 200, 200\}$ in both cases (A), and (B). For RF we set 500 trees, and parametrized the algorithm with $mtry = 3$. All these settings were considered to be good working and obtained from some preliminary runs. In case (A) the results are averaged and the standard deviations out of 10 runs were added. In case (B) we performed likewise 10 runs, but the test set was fixed in this case to the data of unseen persons (a variant comparable to the holdout set error). With the enhancement of parametric bootstrap the resulting CV test set errors from SFA+PB and RF are similar. With the constructed feature set they are considerably better than a Gaussian classifier trained on the original features, which supports the benefits of SFA.

In the evaluation of the unseen persons, only one cross-validation index (defined by the person ID) is present. Here, the standard deviation is determined with respect to the
Table 3.10: Best cross-validation test set error rates for the three algorithms SFA+PB, RF and Gaussian classifier.

<table>
<thead>
<tr>
<th>Settings</th>
<th>SFA+PB</th>
<th>CV test set error</th>
<th>RF</th>
<th>Gaussian</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A) random sampling (716 records)</td>
<td></td>
<td>(2.9±0.3)%</td>
<td>(2.1±0.3)%</td>
<td>(14.0±0.6)%</td>
</tr>
<tr>
<td>(B) unseen persons (716 records)</td>
<td></td>
<td>(15.3±14.9)%</td>
<td>(14.8±15.3)%</td>
<td>(19.6±11.3)%</td>
</tr>
</tbody>
</table>

Table 3.11: CPU times for SFA+PB, RF and a Hidden Markov model [213]. \(N\) is the number of records for each task. The CPU times were obtained on a standard desktop computer, namely an Intel® Core2 Duo CPU T7700 running at 2.4GHz on Windows OS. Best values (shortest CPU time) are written in bold font.

<table>
<thead>
<tr>
<th>Task</th>
<th>(N)</th>
<th>CPU time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(n_{pp} =12)</td>
</tr>
<tr>
<td>train</td>
<td>74</td>
<td>0.55</td>
</tr>
<tr>
<td>train</td>
<td>716</td>
<td>5.60</td>
</tr>
<tr>
<td>apply</td>
<td>64500</td>
<td>6.61</td>
</tr>
</tbody>
</table>

different persons. It is very large (approximately as large as the CV error itself), because there is a large inter-person variation as already described in connection with Fig. 3.11.

But the classification can benefit from a parametric bootstrap enhancement [151] which was originally designed when few training patterns are available, making the SFA algorithm infeasible. If we run classification on unseen gestures without bootstrap (\(N_{copies} = 0\) instead of 200) we get a CV error rate of 17.7% instead of 15.3%. We assume that the variation introduced by the noisy copies of the parametric bootstrap is beneficial for the generalization to unseen persons.

We compared our results with another gesture recognition engine in order to exclude that our gesture set is perhaps “easier” than others. The Hidden Markov Model [213] is included in the software library Wiigee [195] and we used it on a small set of our gestures (74 patterns, randomly selected). The error rate on this training set was 26% which has to be compared with the RF training set error of about 17% from Fig. 3.12(b) for the same training patterns (and with 15% for SFA+PB). Other tests with larger training sets or on unseen test patterns are planned in the future. We conclude however that our gestures do not seem to be fundamentally “easier” for other classifiers. As another comparison we performed the “same-person CV error”-task of [213] on our data and with our SFA+PB model and got a \((2.3 \pm 1.5)\%\) CV error rate where [213] reports 10%.

In Tab. 3.11 we show the CPU times for SFA and RF. Both, RF and SFA, are quite fast.
algorithms, since the training takes only about 0.5 seconds for 74 records and 5 seconds for 716 records. This is faster by a factor of 100 compared with the Hidden Markov recognition engine in the Wiigee package [195], which we tested with the same 74 patterns. The training time for the Gaussian classifier is neglectable, actually it is contained in the reported SFA time. The times for SFA include the extra 200 parametric bootstrap records. They are based on an unoptimized Matlab implementation, so there might be possible improvements. Since a trained SFA model has a very simple structure, we found that applying a trained model to new gestures (last line in Tab. 3.11) is for SFA 3–6 times faster than for RF.

We applied Slow Feature Analysis (SFA) to a time series classification problem originating from gesture recognition. The feature construction by SFA improved the Gaussian classifier, which is included in the Matlab toolkit, by a factor of 2–6. Koch et al. [151] discovered, that small training sets can lead to overfitting by SFA, which could not be observed with larger training data. For this reason they propose an enhancement to SFA for the case of marginal data, which is based on parametric bootstrapping. Summarizing, SFA can be used as a powerful feature construction algorithm for time series problems. Koch et al. [151] obtained results which were equal to or better than the results of a RF classifier, which is really a notable result affirming research question Q1. Especially the fast training and prediction times of SFA have to be highlighted here, which could not be beaten by RF as demonstrated in the runtime comparison. This is an indicator that SFA can be profitably applied for feature pre-processing, and can be seen as a promising alternative to linear methods like PCA. The prediction of unseen test persons appeared to be more difficult for all learning algorithms. Although SFA did not perform much worse than RF and SVM, this task needs further experimentation, partially affirming research question Q2.

3.3 Conclusions

We highlighted the influence of features on the prediction performance of learning algorithms. Features can be the input attributes of the data, or can be constructed based on the original input features, e.g., by using any projection or aggregation method.

One of the most famous feature processing methods is probably feature selection. Feature selection can be used for reducing the number of input attributes of large data sets. Furthermore it is a nice solution to remove irrelevant or redundant information in the data. From a more target-oriented perspective, feature selection aims at increasing the accuracy of learning algorithms. We described differences between methods which are fast to evaluate, that is the filter approaches, and more expensive wrapper approaches. Here the user has to decide how much time can be spent for feature selection, and how good the feature set should be. Wrapper methods sometimes provide a better feature set, but are also more expensive, because the learning process has to be applied several times. Here, filters can be helpful, which only perform a short analysis of the data. However, filters are
3. Feature processing

sometimes not competitive with the more expensive wrappers, and another advantage is that wrappers select features according to the result of a specific learning algorithm. This can be advantageous, especially when the learning algorithm is deteriorated by misleading features.

Another important step in creating good performing learning algorithms is feature construction. Sometimes the input attributes are not appropriate for predicting the target, but combinations or transformations of the input attributes can lead to better features. Therefore, many methods are available in machine learning, e.g., PCA as a representative for linear projections. Creating the monomials of input attributes constitutes a very simple feature projection, which is sometimes already sufficient to improve the prediction. The main disadvantage of monomials is, that the dimensionality of the feature space grows exponentially with an increase in the number of input attributes. For this reason, seldom monomials are built for all input attributes. Here, ranking procedures of feature selection can help to decrease the total number of features generated by the method.

The Slow Feature Analysis (SFA) can generate more sophisticated features compared with rather simple strategies like PCA and monomials. SFA emerged as a method from neuroscience and was originally developed in context of unsupervised learning [254]. We used SFA as pre-processing method for a time series classification task. Here, SFA could achieve comparable performance like state-of-the-art learning algorithms as random forest (RF) or Support Vector Machines (SVMs) but in much shorter time. As a new aspect we discovered, that the size of the training set must be larger than the dimension of the expanded function space of SFA given by a hyperparameter $xpdim$. Otherwise we observed a negative behaviour of the method, resulting in overfitting to the data, that is worse generalization performance. As soon as additional patterns were added by using a resampling technique from statistics, the concept could be learned again. As a consequence we recommended two approaches to circumvent this issue:

i) adding artificial training patterns by using bootstrapped data or
ii) decreasing the expanded function space of SFA, whose dimension is $xpdim$.

In general both options should work well. However, the first option should be preferred if possible, because a decreased function space governed by $xpdim$ can lead to significantly worse accuracy rates. That is, the advantage of SFA compared with PCA or other linear techniques is lost by decreasing the feature space dimension.

The experiments on stormwater prediction showed that a tuning of the pre-processing parameters (here, the two different leaky rain embeddings) and the model parameters at the same time leads to the best performing model. So far, tasks for time series regression were mainly solved by using standard techniques like ARIMA or GARCH [31]. We showed how feature processing can handle time series data as a normal regression task. The pre-processing operator used is generic and can be re-used for similar problems. The crucial
technique responsible for the good results is termed embedding, and consists of adding earlier values of the time series as new features. By doing this, the time dependency of the records is relaxed, which makes it possible to apply any supervised learning algorithm instead of being restricted to specific time series models. A learning algorithm based on SVR could learn the target function very well, using tuning via SPO and the embedding operator for pre-processing only. It showed that the results were a little bit overfitted to the training data, which requires further analysis.

Concluding the feature processing we can say that also combinations of feature projections and feature selection can be helpful in practice. E.g., after applying a PCA, a feature selection of the principal components with the highest variance can reduce the feature number, without losing much information. When simple feature projections do not give sufficient improvements, more powerful non-linear techniques can be helpful. Here we have proposed to use techniques like Genetic Programming or Slow Feature Analysis which can be adapted to arbitrary complexities.