Chapter 7
Predictive Modeling Without Notion of Time
Predictive modeling without notion of time

• Ok, let us start learning
• We will consider learning algorithms that do not take time into account explicitly
  – Of course we do have the temporal features we identified before
• We first need to consider the learning setup
  – Do we want to learn individual models?
  – Do we want to learn to predict well for the future (only for temporal data), or for unseen individuals?
Learning setup

- Here are the options we have (validation set done in the same way):

<table>
<thead>
<tr>
<th>Level</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>individual</td>
<td>( X_{\text{train},qs_i} = x_{1,qs_i}, \ldots, x_{n_{\text{train}},qs_i} )</td>
</tr>
<tr>
<td></td>
<td>( X_{\text{test},qs_i} = x_{n_{\text{train}}+1,qs_i}, \ldots, x_{N_{qs_i},qs_i} )</td>
</tr>
<tr>
<td>population</td>
<td>( X_{\text{train}} \subset { X_{qs_1}, \ldots, X_{qs_n} } ) where (</td>
</tr>
<tr>
<td>unknown users</td>
<td>( X_{\text{test}} \subset { X_{qs_1}, \ldots, X_{qs_n} } ) where (</td>
</tr>
<tr>
<td>population</td>
<td>( X_{\text{train}} = { X_{\text{train},qs_1}, \ldots, X_{\text{train},qs_n} } ) (Note that we refer to the definition of the training sets specified in the row for the individual level here)</td>
</tr>
<tr>
<td>unseen data of known users</td>
<td>( X_{\text{test}} = { X_{\text{test},qs_1}, \ldots, X_{\text{test},qs_n} } )</td>
</tr>
</tbody>
</table>
Learning algorithms

• Assume you know the basic learning algorithms
• Quick recap, two slides per learning algorithm
  – Neural Networks
  – Support Vector Machines
  – K-Nearest Neighbor
  – Decision trees
  – Naïve Bayes
  – Ensembles
Neural Networks (1)

- **Perceptron**

  ![Perceptron Diagram](image)

- **Multi-Layer Perceptron**

  ![Multi-Layer Perceptron Diagram](image)
• Convolutional neural networks
  – Extract features using:
    • Convolutional layers
    • Pooling layers
  – Followed by a “regular” neural network

Neural Networks (2)
Support Vector Machines (1)

- Find hyperplane that maximizes separation between classes

![Diagram of a support vector machine with a hyperplane separating two classes](image)

Fig. 7.6: Example dataset SVMs
Support Vector Machines (2)

- Kernel functions

---

**Kernel functions**

A complex pattern-classification problem, cast in a high-dimensional space nonlinearly is more likely to be linearly separable than in a low dimensional space, provided that the space is not densely populated.

Let us look at an example, shown on the left side of Figure 7.7. This concerns accelerometer data again and we need to classify whether the person is walking or not. Clearly we cannot separate these classes in a linear fashion. Take the following function (referred to as a radial-basis kernel):

\[
K(x, x_0) = e^{-\frac{|x - x_0|^2}{s^2}}
\]

This function is a distance metric between two points. Assume that we have a fixed value for \(x_0\), namely the center of the circles expressed on the left side of Figure 7.7. We calculate the distance of each of our datapoint \(x\) to \(x_0\), which we for now call \(X_3\). The resulting values (combined with the prior values for \(X_1\) and \(X_2\)) are shown on the right side of Figure 7.7. We can now fit a hyperplane that separates the classes by means of the newly introduced values: higher values for \(X_3\) are other activities while lower values indicate walking. This is caused by the points represented in blue (walking) all lie closer to the center (our center point \(x_0\)) compared to all points in...
K-Nearest Neighbor (1)

• Find k closest examples
  – use distance functions discussed before

\[ d(v, g_i) = \begin{cases} 
1 & \text{if } v = g_i \\
0 & \text{otherwise} 
\end{cases} \] (7.14)

When there is a tie we can arbitrarily select a class. In Figure 7.8 we consider our previous example and a new point we have to classify (the triangle). If we select \( k = 3 \) neighbors the points indicated by the yellow star are the 3 nearest neighbors given a Euclidian distance function. Since the majority is blue (i.e. the inactive class), this is the assigned class.

Fig. 7.8: Example k-nearest neighbors

For a regression problem, the computation is even more straightforward. We simply take the average of the values \( y_1, \ldots, y_k \) of the \( k \) nearest neighbors:

\[ \hat{y} = \frac{1}{k} \sum_{i=1}^{k} y_i \] (7.15)

One aspect we have not taken into account is the distance of a neighbor to the point we want to determine the value for. Closer points might be considered more important as they are more similar. This is the basic premise of distance weighted nearest neighbor. We can redefine the equations we have seen for classification and regression by assigning a weight \( w_i \) to each case:
K-Nearest Neighbor (2)

• Assign classes based on some function, for example:
  – Majority class
  – Average value
• Create a tree to decide on target value

Decision tree (1)

![Decision Tree Diagram](image-url)
Decision tree (2)

- Use criterion to decide on which attributes are most important (i.e., should be selected first)

  - Categorical target:

    \[
    \text{entropy}(G) = \sum_{v \in G} -p_v \log_2 p_v \quad \text{where} \quad p_v = \frac{|\{g_i \in G| g_i = v\}|}{|G|}
    \]

    \[
    \text{gain}(X_i, X, G) = \text{entropy}(G) - \sum_{v \in \mathcal{X}_i} \frac{|\{x_j \in X|x^i_j = v\}|}{|X|} \cdot \text{entropy}(\{g_i \in G|x^i_j = v\})
    \]

  - Numerical target:

    \[
    \text{sd\_red}(X_i, X, Y) = \sigma_Y - \sum_{v \in \mathcal{X}_i} \frac{|\{y_i \in Y|x^i_j = v\}|}{|Y|} \cdot \frac{\sigma_{\{y_i \in Y|x^i_j = v\}}}{\sigma_Y}
    \]
Naïve Bayes

- We use Bayes formula to express the probability of a target value $g$ given our observations $x$:

$$p(G = g | x) = \frac{p(x | G = g)p(G = g)}{p(x)}$$

Well, there you go. The class we assign is the one with the highest probability:

$$\hat{g} = \text{argmax}_g p(G = g | x)$$

A disadvantage of the approach we have just discussed is that the assigned probability of a target drops to zero the very instant that we have an attribute-value pair we have not seen in combination with the target. This might be a bit too harsh as it could for example be caused by a lack of sufficient training data. The Laplace estimator can help here. We simply add 1 to the numerator and the number of values of the attribute to the denominator, i.e.

$$p(X_i = v | G = g) = \frac{1 + \left| \{j \in \{2, \ldots, N\} | (x_i j = v)^g_i = g \} \right|}{\left| X_i \right| + \left| \{j \in \{2, \ldots, N\} | g_j = g \} \right|}$$

This gives an equal probability in case we have no observations, and moves to the actual observed probably in the dataset when $N$ becomes large. If a value for a certain attribute is missing we simply ignore that attribute in the calculations, a very natural and simple way of handling missing values. This is one of the advantages of the Naïve Bayes classifier.

So far we have only considered categorical attributes. Naïve Bayes is able to cope with numerical values as well, and in fact it does so in a very natural way. We can create a probability density function to represent $p(X_i = v | G = g)$ and derive the probability from that function. A typical example is a probability density function in the form of normal distribution $N(\mu, \sigma)$ where we use the maximum likelihood to estimate $\mu$ and $\sigma$ from our data. It can be proven that the best values would be the mean and standard deviation observed in the data (i.e. the observations in the set $\{x_j \in X | g_i = g\}$).

7.7 Ensembles

It is time to kick it up a notch. We have seen a variety of relatively simple models. We hate to admit it, but we could suffer from a lack of performance using these approaches. There can be two main causes of this: (1) the expressive power of the models might be insufficient causing a hampered performed (this cause is called the bias of the model), or (2) the training data is simply too limited (variance of the data). Ensembles can help out here. The idea behind ensemble learning is not to create a single model using the algorithms we have considered before, but a number of these models. In the end, the ensemble of models returns a combined answer.
Ensembles (1)

- Bagging
  - Create multiple models on samples of the data

\begin{algorithm}
models = []
for i in 1,...,m do
  Take a sample \( X_i \subset X \)
  Build a model \( M_i \) on training data \( X_i \) using some learning algorithm
  Add \( M_i \) to models
end
\end{algorithm}

- Combine output of models using some aggregation function
- Tackles variance problem
Ensembles (2)

• Boosting
  – Build models sequentially and focus models on where we made mistakes before
  – Focuses on bias problem

\[ X_1 = X \]
\[ X_2 \text{ (weighted)} \]
\[ X_m \]

\[ M_1 \]
\[ M_2 \]
\[ M_m \]
\[ M_{\text{combined}} \]

Let us take a closer look into how we can compose the new training sets. We will consider an example of a boosting approach aimed at classification first. It is called Arcing ([45]). Arcing works on the principle of assigning sampling probabilities \( p_1, \ldots, p_N \) to each of our \( N \) training elements. As said, we will build \( m \) models (i.e. take \( m \) steps), and in the \( k \)th step the probabilities (with the identification of the step in brackets, e.g. \( p_i(k) \)) are set based upon the error of the ensemble we have built so far. Let \( d_i \) denote whether the training example \( x_i \) is incorrectly classified with the current model \( k \) (\( d_i = 1 \)) or not (\( d_i = 0 \)). We define the weighted misclassification rate \( e_k \) in step \( k \) as follows:

\[ e_k = \sum_{i=1}^{N} p_i(k) d_i \quad (7.31) \]

The factor \( b_k \) is defined to compute the eventual new weights. It is defined in the equation below.

\[ b_k = \frac{1}{e_k} \quad (7.32) \]

The higher the weighted misclassification rate \( e_k \) the lower \( b_k \). The algorithm assumes \( e_k \leq \frac{1}{2} \) otherwise it terminates. The termination criterion essentially says that the execution stops when you are predicting with a higher misclassification rate than a random choice for a balanced binary classification problem. This means that \( b_k \) resides in the interval \([1, \cdot] \) where 1 means the highest misclassification rate (i.e. \( \frac{1}{2} \)).
Practical considerations

• Ok, let us start to apply the learning algorithms!
  – We need to be careful not to overfit (think of the first set of slides)
  – We can select features using forward and backward selection
  – We can consider regularization (punishing more complex models)
**Forward selection**

- We iteratively add the most predictive feature:

```plaintext
Algorithm 17: Forward selection

selected_attributes = {}
performances = []
for k = 1, ..., p do
    best_attribute = ""
    best_performance = ∞
    available_attributes = X \ selected_attributes
    for l ∈ 1, ..., |available_attributes| do
        temp_attributes = selected_attributes ∪ available_attributes_l
        performance = learn_model(temp_attributes, X)
        if performance < best_performance then
            best_attribute = available_attributes_l
            best_performance = performance
        end
    end
    performances[k] = best_performance
    selected_attributes = selected_attributes ∪ best_attribute
end
return performances
```
Backward selection

• We iteratively remove the least predictive feature:

```python
Algorithm 18: Backward selection

selected_attributes = X
performances = []
performances[p] = learn_model(selected_attributes, X)
for k = p - 1, ..., 1 do
    worst_attribute = ""
    best_performance = ∞
    available_attributes = selected_attributes
    for l ∈ 1, ..., |available_attributes| do
        temp_attributes = selected_attributes \ available_attributes_l
        performance = learn_model(temp_attributes, X)
        if performance < best_performance then
            worst_attribute = available_attributes_l
            best_performance = performance
        end
    end
    performances[k] = best_performance
    selected_attributes = selected_attributes \ worst_attribute
end
return performances
```

7.9.2 Regularization

As we have discussed in Chapter 6 we want to learn models that are generalizable, i.e. that allow us to make accurate predicts for unseen data, not just for the training data. It is known that the more complex a model becomes, the more prone it is to overfitting, similar to what we have discussed for feature selection. The concept of regularization is there to help out. We can add a regularization term $R$ to the objective of our learning algorithms. This term typically punishes the complexity of the model that has been found, and will allow us to strike a balance between performance and model complexity during the training process. The higher the value for $R$ the more influence it will have in the learning process. For neural networks an example of a regularization terms is the following:
Regularization

• We add a term to the error function to punish for more complex models
  – For neural networks:

\[
E = \sum_{i=0}^{N} (y_i - \hat{y}_i)^2 + \frac{\lambda}{2} ||w||
\]
CrowdSignals

• Remember, we have two tasks for our CrowdSignals dataset:
  1. Predict label (activity)
  2. Predict heart rate

• We consider the first with a non-temporal setup and the second with a temporal setup
Predicting the label

- We currently have a set of binary attributes for each label value
  - We need to transform these to a single categorical attribute
  - How do we do this? What if we have multiple “1”s per instance?
  - We only use instances where we have a single activity marked with a 1
  - Results in 1837 instances (out of 2895)
Splitting the data

- How do we split the data into training and test?
  - We assign 70% of the instances to the training set and 30% to the test set
  - We do so in a *stratified* way
  - Once again, we ignore the time component
Feature selection (1)

• We have 517 features if we use all the ones we have identified
  – Initial features (21)
  – PCA (7)
  – Time domain features, mean + standard deviation (2 x 28 features = 56)
  – Frequency domain (18 frequencies, 24 initial features = 432)
  – Cluster (1)

• Should we select some features?
Feature selection (2)

- **Forward selection:**

  ![Graph showing accuracy vs number of features]

  First 50 features. We see that after selecting around 5 features the performance on the training set no longer substantially increases (by the way, look at the high accuracy scores we obtain in this initial run already!). We select 10 features as this number of features is selected based on a decision tree, other approaches might require slightly more features because they are able to exploit the additional information.

  The selected features include only time-and frequency-based features. Time-related features that are selected are the mean of the pressure measured by the phone and the standard deviation of the x-axis of the gyroscope. The frequency features included are the amplitude of the 0 Hz frequency for the y-axis of the accelerometer of the phone, the entropy of the y-axis of the magnetometer of the watch, the frequency with the highest amplitude for the z-axis of the magnetometer, the weighted frequency of the y-axis of the gyroscope of the watch, the amplitude of the 1 Hz frequency for the y-axis of the phone's gyroscope, the amplitude of the 1.9 Hz frequency of the x-axis of the accelerometer of the phone, and the amplitude of the 0.9 Hz frequency and 0.5 Hz frequency for the z-axis of the magnetometer of the watch and the y-axis of the accelerometer respectively.
Feature selection (3)

- We end up with different options which we will try:

<table>
<thead>
<tr>
<th>Dataset name</th>
<th>Basic features</th>
<th>PCA</th>
<th>Time-and frequency features</th>
<th>Clusters</th>
<th>#features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial set (cleaned)</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td>21</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>21</td>
<td>7</td>
<td></td>
<td></td>
<td>28</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>21</td>
<td>7</td>
<td>56 + 432</td>
<td></td>
<td>516</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>21</td>
<td>7</td>
<td>56 + 432</td>
<td>1</td>
<td>517</td>
</tr>
<tr>
<td>Selected features</td>
<td></td>
<td></td>
<td>2 + 8</td>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>
Influence of parameter settings (1)

• How does model complexity influence our performance in this case?
  – Remember the theoretical results we have seen
  – We can tailor the parameters of the models
  – We can change the weight of the regularization terms
Influence of parameter settings (2)

- Neural network and regularization term

We have identified our training sets as well as the different subsets of features we want to use. We are almost ready to apply our learning algorithms. Before we apply them however, we should be aware that we do not want to overfit our models towards the training data (remember our discussion in Chapter 6). For this, we can punish overly complex models (regularization) and we should also set the parameters of the learning algorithms in an appropriate way.

Let us first consider regularization. As we have discussed in the Neural Networks explanation we can add a regulizer to our equations, punishing more complex models. The regulizer comes with a regularization parameter. The higher the parameter value, the more complex models are punished, and hence, the more simple models (with low weights in this case) are favored. Figure 7.12 shows the impact of the regularization parameter upon the performance for both the training and the independent test set. Although the differences are not that big (this is very dependent on the variance of the data), we do observe a trend that the performance on the training set goes down when we increase the value of the parameter while the performance on the test set goes up. Hence, we end up with a better generalizable model. We should thus take care to create overly complicated models.

Fig. 7.12: Influence of regularization parameter upon accuracy (notice the log scale on the x-axis)

Next, the setting of the parameter values of the learning algorithms can severely impact their performance. Some represent a notion of complexity as we have seen in our previous example, while others represent different aspects (e.g. the shape of the decision function). We will focus on a decision tree here. One parameter allows us to set the minimum number of examples per leaf. If we set this number too low overfitting will be highly likely to occur since leafs represent only limited examples. Hence, we could say that this parameter says something about the complexity of the tree again. If we consider Figure 7.13 we see that performance on the training set increases when we decrease the minimum number of example per leaf. The same
Influence of parameter settings (3)

- Decision tree and points per leaf:

![Graph showing the accuracy of decision tree with varying minimum number of points per leaf. The x-axis represents the minimum number of points per leaf, ranging from 1 to 10, and the y-axis represents the accuracy, ranging from 0.95 to 1.00. The graph has two lines: one for training and one for test, illustrating the performance drop when a value smaller than 5 is selected, indicating overfitting.]
Influence of parameter settings (4)

- We try different parameter settings
  - study the generalizability based on a cross validation on the training set

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Variant description</th>
<th>Parameters varied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Network (NN)</td>
<td>Multi-Layer Perceptron with 1 or 2 hidden layers, a logistic activation function and one output node per class</td>
<td>hidden layer composition: {single layer with 5, 10, 25, or 100 neurons, two layer with 100 and 5 neurons or 100 and 10} maximum iterations: {1000, 2000}</td>
</tr>
<tr>
<td>Support Vector Machines (SVM)</td>
<td>SVM with kernel function with one SVM model per class</td>
<td>maximum iterations: {1000, 2000} C: {1, 10, 100} tolerance = {0.001, 0.0001} kernel function: {rbf, polynomial}</td>
</tr>
<tr>
<td>K-Nearest Neighbor (KNN)</td>
<td>KNN model using simple Euclidean distance</td>
<td>k: {1, 2, 5, 10}</td>
</tr>
<tr>
<td>Decision Tree (DT)</td>
<td>Decision tree algorithm following the CART approach</td>
<td>minimum samples per leaf: {2, 10, 50, 100, 200} splitting criterion: {gini, entropy}</td>
</tr>
<tr>
<td>Naïve Bayes (NB)</td>
<td>Basic Naïve Bayes approach</td>
<td>-</td>
</tr>
<tr>
<td>Random Forest (RF)</td>
<td>Basic Random forest approach</td>
<td>minimum samples per leaf: {2, 10, 50, 100, 200} number of trees: {10, 50, 100} splitting criterion: {gini, entropy}</td>
</tr>
</tbody>
</table>
Results test set (1)

Fig. 7.14: Visualization of accuracy include confidence intervals

Fig. 7.15: The resulting decision tree using the selected features

Finally, we can explore where our algorithm tends to make mistakes (i.e. which labels are confusing). The confusion matrix is shown in Table 7.16. We observe a lot of high numbers on the diagonal, indicating a high accuracy. We do see that the algorithm predicts a very limited number of cases incorrectly, washing hands is classified as eating two times, while walking is predicted to be standing for two times.
Results test set (2)

• How did we compute the 95% confidence intervals?
  – $a$ is the accuracy
  – $n$ the number of samples in the test set

$$sd = \sqrt{a(1 - a)/n}. $$
Results test set (3)

- Important features (random forest):

<table>
<thead>
<tr>
<th>Feature</th>
<th>Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>gyr_phone_x_temp_std_ws_120</td>
<td>0.2855</td>
</tr>
<tr>
<td>press_phone_pressure_temp_mean_ws_120</td>
<td>0.2499</td>
</tr>
<tr>
<td>acc_phone_y_freq_0.0_Hz_ws_40</td>
<td>0.2433</td>
</tr>
<tr>
<td>mag_watch_y_pse</td>
<td>0.0873</td>
</tr>
<tr>
<td>mag_phone_z_max_freq</td>
<td>0.0341</td>
</tr>
<tr>
<td>gyr_watch_y_freq_weighted</td>
<td>0.0284</td>
</tr>
<tr>
<td>gyr_phone_y_freq_1.0_Hz_ws_40</td>
<td>0.0255</td>
</tr>
<tr>
<td>acc_phone_x_freq_1.9_Hz_ws_40</td>
<td>0.0227</td>
</tr>
<tr>
<td>acc_watch_y_freq_0.5_Hz_ws_40</td>
<td>0.0164</td>
</tr>
<tr>
<td>mag_watch_z_freq_0.9_Hz_ws_40</td>
<td>0.0068</td>
</tr>
</tbody>
</table>

To summarize, we obtain excellent results, but it is hard to estimate whether we would perform equally well if we would have a richer dataset. Still, the steps that we have explained are precisely the same ones as you need to perform for any other dataset, which was the main purpose of this case study of course.
Results test set (4)

- Confusion matrix:

![Confusion Matrix Diagram]

To summarize, we obtain excellent results, but it is hard to estimate whether we would perform equally well if we would have a richer dataset. Still, the steps that we have explained are precisely the same ones as you need to perform for any other dataset, which was the main purpose of this case study of course.
Results test set (5)

• Why are the results this good?
  – Dataset has limited variation
  – Spans only short time period
  – …
Predicting the heart rate

- We take it as a temporal problem
  - Want to learn to predict future time points well
- How do we split the training and test set here?
- Why this way?

![Training and Test Set Split for Heart Rate Prediction Task](image)
# Features

The number of features included. We have a few more basic features (we clearly exclude heart rate, but include the binary attributes representing the individual activity labels). The same holds for the temporal features: the categorical temporal features for the labels have been included while the temporal features associated with the heart rate are no longer present.

<table>
<thead>
<tr>
<th>Dataset name</th>
<th>Basic features</th>
<th>PCA</th>
<th>Temporal features</th>
<th>Clusters</th>
<th>#features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial set (cleaned)</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td>21</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>28</td>
<td>7</td>
<td></td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>28</td>
<td>7</td>
<td>71 + 432</td>
<td></td>
<td>538</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>28</td>
<td>7</td>
<td>71 + 432</td>
<td>1</td>
<td>539</td>
</tr>
<tr>
<td>Selected features</td>
<td>1</td>
<td>1</td>
<td>6 + 2</td>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>

We also made a selection of features, based on the Pearson coefficient. The correlation values for the selected features, which are the 10 features with the most extreme correlations, as shown in Table 7.8. When observing the correlations with the highest magnitudes, we see that the fact that the position of the phone on the table is quite important as are a few principal components. Note that the temporal feature for the phone lying on the table is slightly different from the regular label as it is true if it has been observed within the historical window. We again see that time and frequency based features play an important role.

## Tuning parameters

For tuning parameters we take the same approach as we have considered before, namely to do a grid search over parameter values that are potentially suitable. We again use a cross validation scheme since our learning algorithms do not care about the order anyway (this will change in the next Chapter). An overview of the algorithms we use (again, all of those that we have explained and can be used for solving a regression problem have been included) and the parameter values that are part of the grid search is shown in Table 7.9. We use support vector regression without a kernel function due to the lengthy runtimes. We again average the performance of the random forest and the neural network over five runs.
## Algorithms we tried

### Table 7.8: Ten most important features using Pearson correlation

<table>
<thead>
<tr>
<th>Feature</th>
<th>Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>temp</td>
<td>0.6158</td>
</tr>
<tr>
<td>pattern</td>
<td>0.6158</td>
</tr>
<tr>
<td>labelOnTable</td>
<td>0.6158</td>
</tr>
<tr>
<td>pca</td>
<td>0.5265</td>
</tr>
<tr>
<td>temp</td>
<td>0.5221</td>
</tr>
<tr>
<td>pca</td>
<td>0.5022</td>
</tr>
<tr>
<td>acc</td>
<td>0.5022</td>
</tr>
<tr>
<td>gyr</td>
<td>-0.5368</td>
</tr>
<tr>
<td>gyr</td>
<td>-0.6419</td>
</tr>
</tbody>
</table>

### Table 7.9: Algorithms variant and parameters varied for the regression problem

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Variant description</th>
<th>Parameters varied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Network (NN)</td>
<td>Mutli-Layer Perceptron with 1 or 2 hidden layers and an identity activation function and one output node per class</td>
<td>hidden layer composition: {single layer with 5, 10, 25, or 100 neurons, two layer with 100 and 5 neurons or 100 and 10} maximum iterations: {1000, 2000}</td>
</tr>
<tr>
<td>Support Vector Regression (SVR)</td>
<td>Linear SVR (no kernel function)</td>
<td>maximum iterations: {1000, 2000} C: {1, 10, 100} tolerance = {0.001, 0.0001}</td>
</tr>
<tr>
<td>K-Nearest Neighbor (KNN)</td>
<td>KNN model using simple euclidean distance</td>
<td>k: {1, 2, 5, 10}</td>
</tr>
<tr>
<td>Decision Tree (DT)</td>
<td>Decision tree algorithm following the CART approach with the mean squared error as a splitting criterion</td>
<td>minimum samples per leaf: {50, 100, 200}</td>
</tr>
<tr>
<td>Random Forest (RF)</td>
<td>Basic Random forest approach with the mean squared error as a splitting criterion</td>
<td>minimum samples per leaf: {50, 100, 200} number of trees: {10, 50, 100}</td>
</tr>
</tbody>
</table>
Results (1)

<table>
<thead>
<tr>
<th>Approach</th>
<th>NN (training)</th>
<th>NN (test)</th>
<th>RF (training)</th>
<th>RF (test)</th>
<th>SVM (training)</th>
<th>SVM (test)</th>
<th>KNN (training)</th>
<th>KNN (test)</th>
<th>DT (training)</th>
<th>DT (test)</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial set</td>
<td>725.3 (1009.3)</td>
<td>1584.6 (1417.6)</td>
<td>582.5 (857.1)</td>
<td>1548.0 (1346.5)</td>
<td>1155.8 (1216.8)</td>
<td>1303.3 (1067.2)</td>
<td>186.8 (574.1)</td>
<td>2309.7 (2027.9)</td>
<td>542.8 (855.8)</td>
<td>1496.7 (1268.4)</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>716.3 (971.6)</td>
<td>1510.0 (1333.6)</td>
<td>476.0 (800.9)</td>
<td>1750.4 (1500.1)</td>
<td>1600.2 (4785.8)</td>
<td>1465.2 (1609.4)</td>
<td>186.8 (574.1)</td>
<td>2309.7 (2027.9)</td>
<td>537.2 (850.8)</td>
<td>1501.4 (1282.2)</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>523.6 (846.2)</td>
<td>1968.0 (2801.3)</td>
<td>15.1 (41.6)</td>
<td>1310.3 (1129.2)</td>
<td>1516.1 (2783.7)</td>
<td>1967.4 (3950.7)</td>
<td>134.2 (473.6)</td>
<td>2259.0 (1848.2)</td>
<td>368.2 (643.8)</td>
<td>1317.0 (826.8)</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>557.7 (861.5)</td>
<td>2014.7 (2114.9)</td>
<td>7.0 (21.1)</td>
<td>1384.8 (1292.9)</td>
<td>1211.8 (2051.6)</td>
<td>1997.2 (3243.5)</td>
<td>134.2 (473.6)</td>
<td>2259.0 (1848.2)</td>
<td>368.2 (643.8)</td>
<td>1317.0 (826.8)</td>
</tr>
<tr>
<td>Selected features</td>
<td>768.9 (1369.2)</td>
<td>2825.3 (2855.4)</td>
<td>620.7 (1109.3)</td>
<td>2614.8 (2613.4)</td>
<td>808.5 (1909.1)</td>
<td>2598.2 (2944.2)</td>
<td>286.1 (719.9)</td>
<td>2571.1 (2896.0)</td>
<td>538.6 (1039.8)</td>
<td>2740.0 (2878.4)</td>
</tr>
</tbody>
</table>
Results (2)

- Standard deviation computed based on predictions per instance on the test set
- Predictors random forest:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>acc_watch_y_temp_mean_ws_120</td>
<td>0.4631</td>
</tr>
<tr>
<td>press_phone_pressure_temp_mean_ws_120</td>
<td>0.1398</td>
</tr>
<tr>
<td>mag_watch_x_temp_mean_ws_120</td>
<td>0.0465</td>
</tr>
<tr>
<td>temp_pattern_labelEating</td>
<td>0.0404</td>
</tr>
<tr>
<td>mag_watch_y_temp_mean_ws_120</td>
<td>0.0385</td>
</tr>
<tr>
<td>temp_pattern_labelEating</td>
<td>0.0383</td>
</tr>
<tr>
<td>labelEating</td>
<td>0.0382</td>
</tr>
<tr>
<td>pca_4_temp_mean_ws_120</td>
<td>0.0208</td>
</tr>
<tr>
<td>pca_7</td>
<td>0.0131</td>
</tr>
<tr>
<td>pca_7_temp_mean_ws_120</td>
<td>0.0112</td>
</tr>
</tbody>
</table>
7.11 Exercises 161

1. Imagine that we have a dataset with instances covering measurements that have been obtained using different mobile phones. What would your learning setup be? Will you just consider the dataset as a whole, or would you split it up? Take the learning setups we have considered into account.

2. Give an overview of the complexity of the following learning algorithms we have presented in this chapter: neural networks, support vector machines (with and without kernel), k-nearest neighbor, decision trees, and naive bayes. Do this for the training process for a classification problem and express the complexity in terms of the number of instances $N$ and the number of features $p$.

3. Setting the parameter values of the learning algorithms properly is an important step to obtain good results (we have seen that in our case study), but trying out a lot of values might be very time consuming. Sometimes some rough guidelines are however available. Try to find a guideline on the number of neurons to use for a simple multi-layer perceptron network.

4. While back propagation is a very popular algorithm to train a neural network, it does come with some severe problems. List two disadvantages of the algorithm and provide a solution to each of these problems from the literature.

5. It is known that normalization of the input and output of the neural network is generally a good idea (although we did not fully do it in this case), explain why.

6. In our explanation of support vector machines we mentioned the existence of kernel functions. Also in deep neural networks and other approaches these are commonly seen. List at least three kernel functions and explain their properties.
Why is it so bad?

• Ideas?