Capstone Project **Machine Learning Engineer Nanodegree**	Sebastian Nietardt January 16, 2017

Human Activity Recognition with Smartphones

Definition

Project Overview

Most people, nowadays, own a smartphone and carry it close to their body most of the time. A smartphone has several sensors to measure its movement. This sensor values can be used for guessing what the user is doing. This information can be valuable in several situations. One can use it to guess the calories burned by knowing how many steps its owner took, how fast the steps were taken and if a staircase or a slope was involved. To name another example, I could imagine the data could also be used by the police or insurance companies to figure out if its owner was using the smartphone while driving. In this project, we will try to use the sensor data of the accelerometer and gyroscope to guess what kind of activity the smartphone's owner is doing. The six activities which we are trying to guess are WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING and LAYING.

The dataset was obtained from kaggle. It has been released under the CC0: Public Domain License and can therefore be used without any restrictions. The dataset has been provided by UCI Machine Learning.

The complete description of the dataset can be found on kaggle. The following paragraph is a short summary of the description which UCI Machine Learning provided together with the dataset. Link to the description and the dataset.

30 volunteer subjects participated in this project. The data is already split into training data (containing the data of 21 subjects) and testing data (containing the data of 9 subjects). The signals of two sensors (accelerometer and gyroscope) have already been pre-processed by applying noise filters and have been sliced into 2.56 second intervals with 50% overlap. The gravitational and body motion components of the acceleration signal where separated, into body acceleration and gravity, using a Butterworth low-pass filter. The final dataset consists of 561 features which were obtained by calculating variables from each interval. The training data contains 7,352 examples and the testing data contains 2,947 examples. Each example has an activity as well as an anonymous subject assigned to it.

Problem Statement

This project is about recognizing human behavior with a smartphone. A smartphone is constantly measuring its movement and when its owner is carrying it, these measurements can be used for guessing its owner's activity. In this project, we have six activities which we are trying to predict. These activities are WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING and LAYING. This problem is quantifiable, since it has a lot of sensor data which are expressed in numbers and exactly one activity assigned to it. It is measurable, because we have a defined outcome and data to train and test on. And it is replicable, because it can be reproduced with every smartphone.

In this project, we will apply some algorithms to the dataset and will find out which works best. After a closer look at the data, the first step will be to do feature scaling and to remove some outliers if it increases the predicting power of the models. The second step will be feature transformation to emphasize important features. In the third step, a classifier will be applied to the transformed data which will learn to predict the activity of a subject by the provided examples. This is called supervised learning. The fourth and last step will be tuning. In this step, the best model will be applied to the dataset several times using different parameters. The model with the best performance will be our final model for this project.

The anticipated solution is a model, which can recognize the six activities WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING and LAYING just by the values of the input features from the sensors. It should be able to do this guesses not just for the training data but also for any data from any human subject it has not seen before, as long as the unseen data is in same the shape as the training data.

Metrics

The problem has exactly six outcomes which the algorithms are trying to predict. There are no close answers. Each prediction is either right or wrong. This means the best way to evaluate the predictive power of the model is an accuracy score. The accuracy score counts all the right predictions and divides the result by the total amount of predictions made. This calculation leads to a number between 0 and 1 which expresses the percentage of correct predictions.

Example:

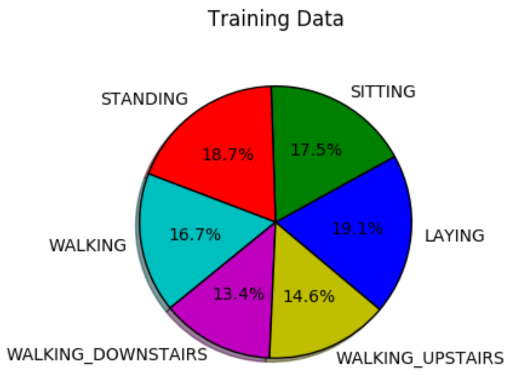
Total predictions:	20
Correct predictions:	17
Accuracy Score:	17 / 20 = 0.85 = 85 %

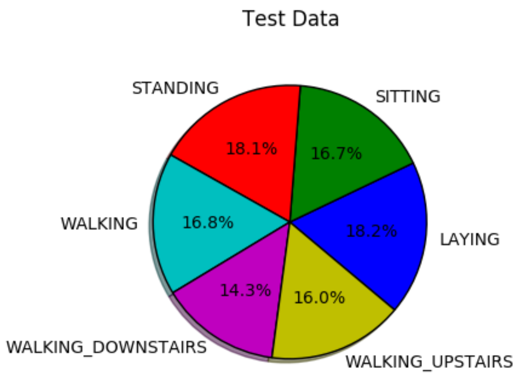
Another good metric for the project would be the f1 score which is a combination of the precision score and the recall score. The f1 score will also be calculated and visualized in the conclusion section because it is a good indicator for robustness and shows more details about the model's performance. The reason the accuracy score was chosen over the f1 score is that it is simple to understand. People without machine learning background do not have to read the whole paper or to know about precision and recall to look at the result and get an impression of the model's predictive power.

Analysis

Data Exploration

There are 7,352 examples in the training dataset. Each example in the dataset has 561 features, a subject and an activity. First, we need to know how many examples we have for each activity. This is important because we need enough examples for each activity for the model to train on. The following pie chart shows the number of examples in percent within the training dataset and the testing dataset. It looks like the dataset is well prepared. It has about the same number of examples for each activity.



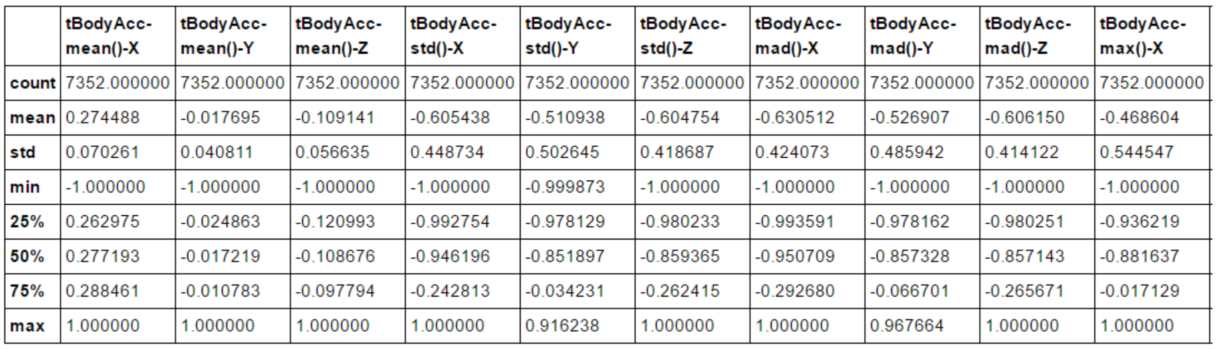


The subject must be dropped from the dataset. This is important, because the test subjects from the training dataset will neither appear in the testing dataset nor do we want to limit the final model to any subjects.

Now let us have a closer look at the head of the first ten features. Examples of it are shown in the following table. It looks like features next to each other have often similar values or might even correlate. The same preprocessed sensor data seems to be divided into several columns.

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-0.923527	-0.934724
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-0.957686	-0.943068
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-0.977469	-0.938692
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750	-0.989302	-0.938692
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672	-0.990441	-0.942469

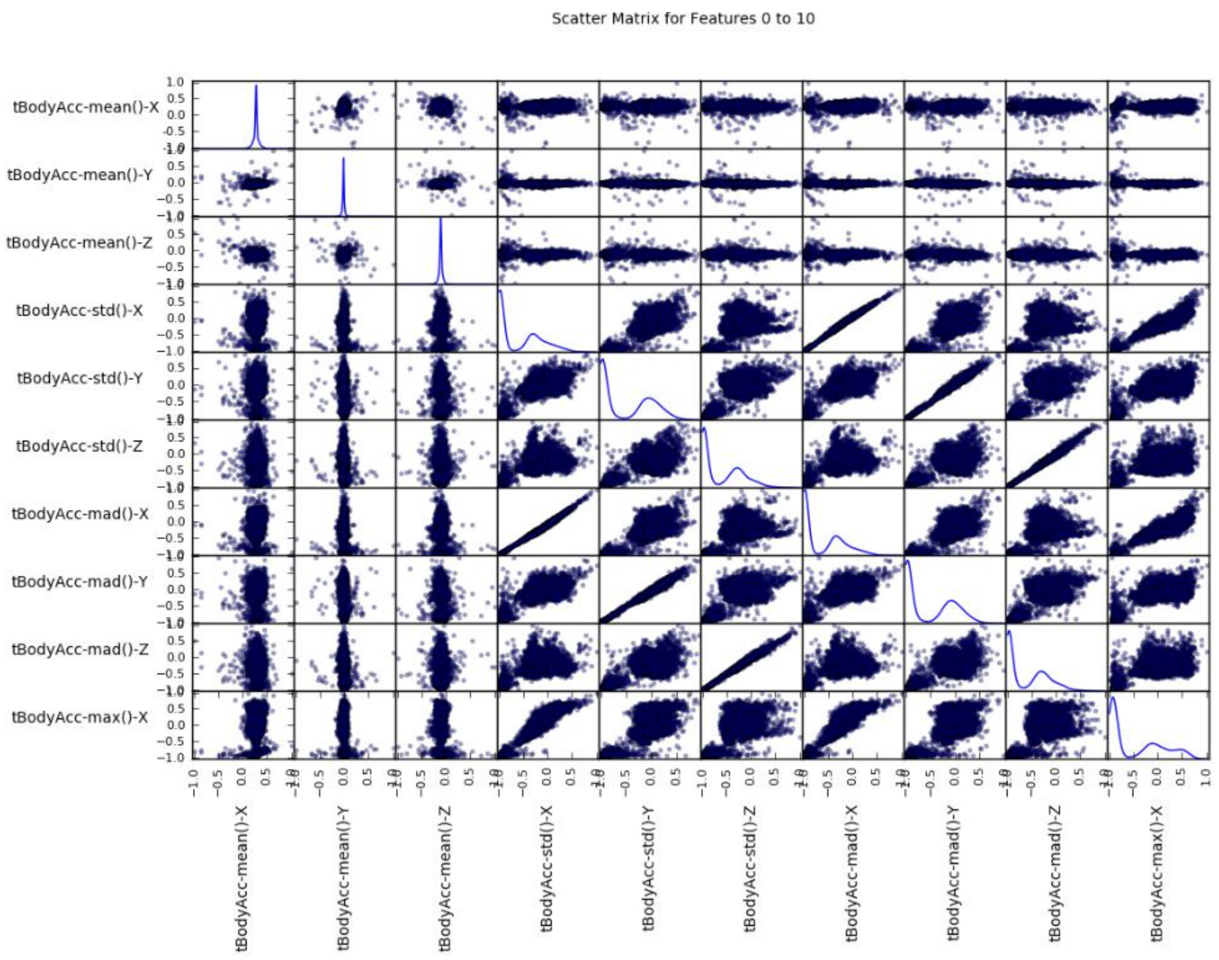
There are no categorical variables in the dataset, except the activity labels and the subjects which were already removed from the dataset. Fortunately, there are no missing values either. The values of each feature range from minus one to plus one and the data type of each feature is float64. The means, medians, modes, standard deviation, upper quartiles and lower quartiles vary a lot. The following table shows a description of the first ten features.



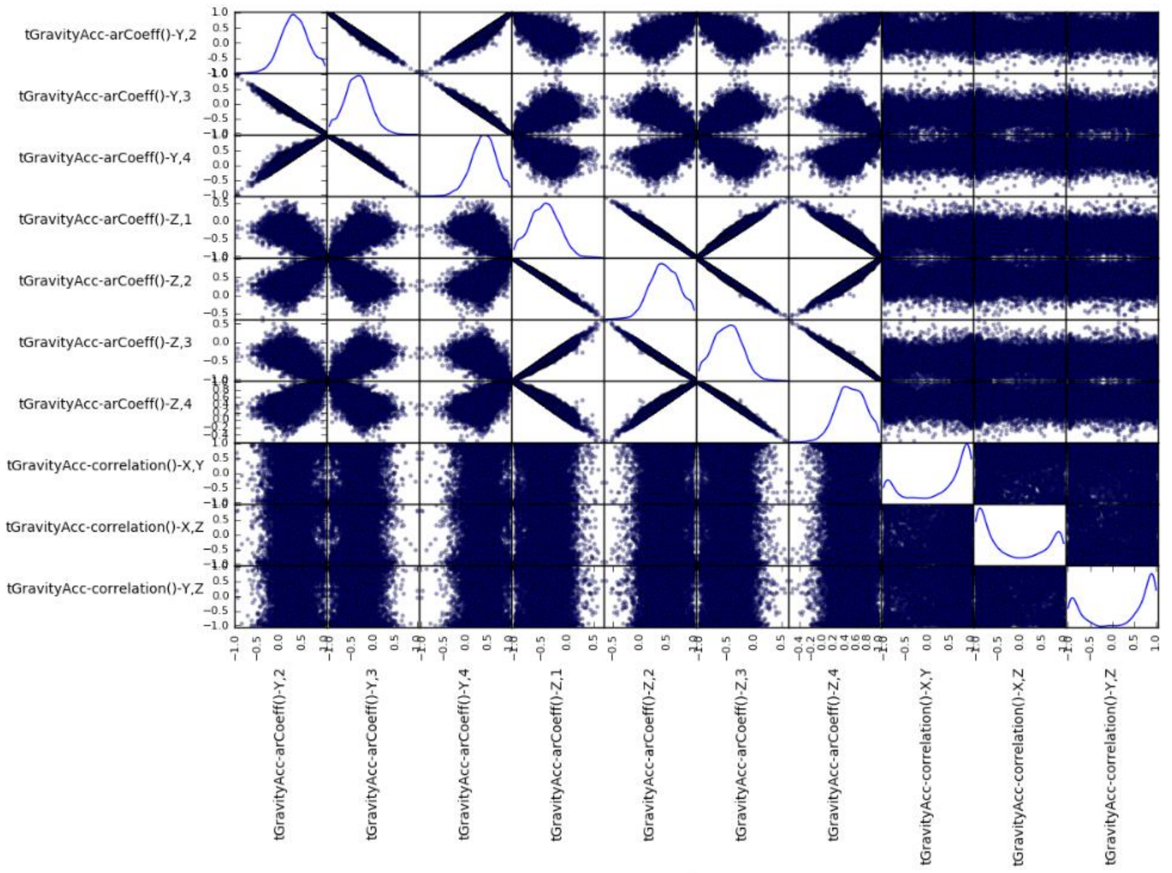
	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X
count	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000	7352.000000
mean	0.274488	-0.017695	-0.109141	-0.605438	-0.510938	-0.604754	-0.630512	-0.526907	-0.606150	-0.468604
std	0.070261	0.040811	0.056635	0.448734	0.502645	0.418687	0.424073	0.485942	0.414122	0.544547
min	-1.000000	-1.000000	-1.000000	-1.000000	-0.999873	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000
25%	0.262975	-0.024863	-0.120993	-0.992754	-0.978129	-0.980233	-0.993591	-0.978162	-0.980251	-0.936219
50%	0.277193	-0.017219	-0.108676	-0.946196	-0.851897	-0.859365	-0.950709	-0.857328	-0.857143	-0.881637
75%	0.288461	-0.010783	-0.097794	-0.242813	-0.034231	-0.262415	-0.292680	-0.066701	-0.265671	-0.017129
max	1.000000	1.000000	1.000000	1.000000	0.916238	1.000000	1.000000	0.967664	1.000000	1.000000

Exploratory Visualization

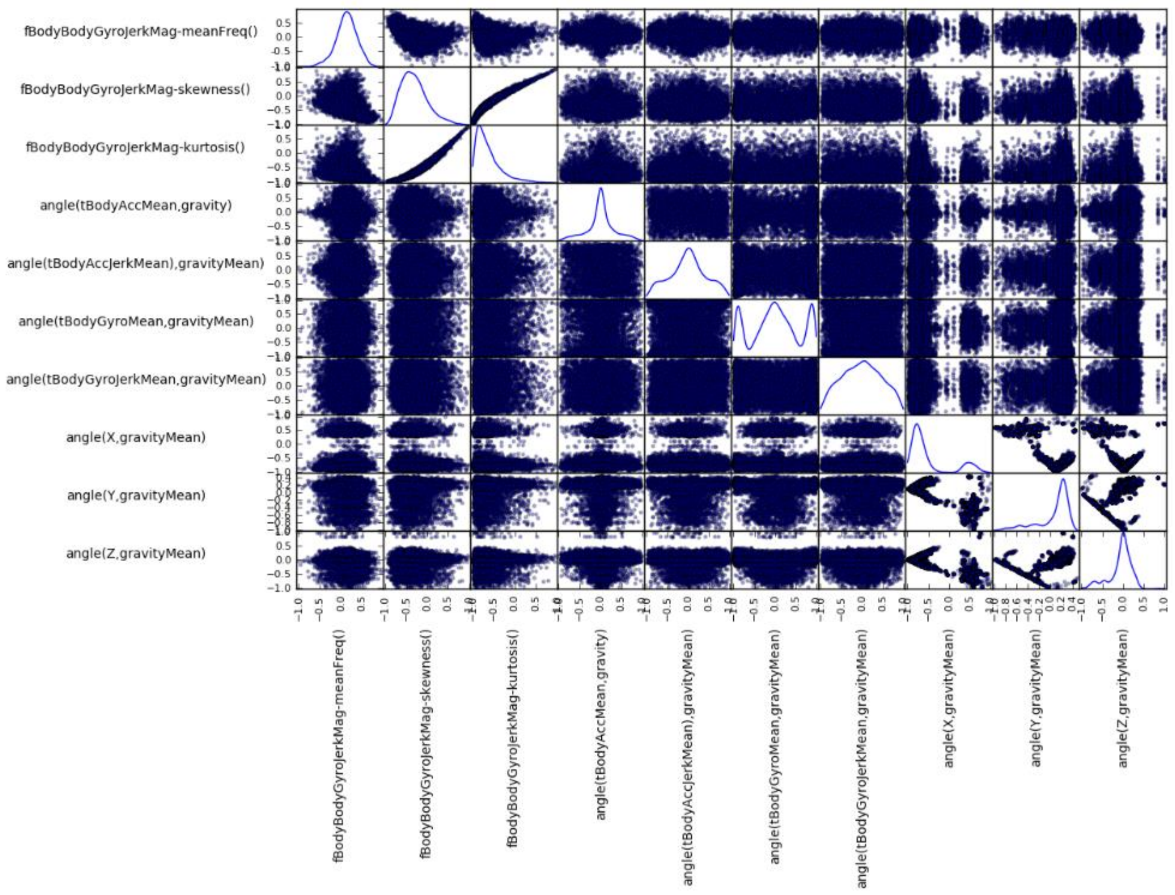
The dataset has too many features to show the correlation between all of them in one correlation matrix. But we will still have a glimpse at the correlation between some of them. By looking at the correlation matrix of the first ten features in the dataset, some features further in the middle and the ten features at the end of the dataset, we can see some correlations between them. On the first and second scatter plot are some nearly straight lines. The columns tBodyAcc-mad()-X and tBodyAcc-std()-X, for example, seem to correlate. Some distributions are negatively skewed others are positively skewed, some are multimodal and some are normally distributed. The data really varies a lot.



Scatter Matrix for Features 70 to 80



Scatter Matrix for Features 551 to 561



Algorithms and Techniques

The solution to the problem can be obtained by several machine learning techniques. Since we have labeled data, we can use a supervised learning algorithm. The output we are trying to predict is discrete. That means it is a classification problem. We will apply the algorithms k-nearest neighbors (KNN), AdaBoost, multi-layer perceptron (MLP), naive Bayes, support vector machines (SVM) and logistic regression to the dataset and see how they perform. All of them are good classifiers for supervised learning classification problems.

K-nearest neighbors stores the feature values for each training example in its model like in a coordinate system. When it has to make a prediction, it finds the k-closest data points in its coordinate system and let them vote for an answer.
AdaBoost uses a series of weak learners (like e.g. a simple decision tree) to make its prediction. After each simple classification, it adapts the weights of the data points. It increased the weight of the wrongly classified data points, decreases the weights of the correctly classified data points and applies the next simple classification. The final prediction is a weighted vote of all simple classifiers.
Multi-layer perceptron is an artificial neural network. It builds up multiple layers of neurons. The neurons in the first layer represent the input features from the dataset. All other neurons receive weighted outputs from the layer before them as inputs and activate as soon as the input value is higher than the activation threshold.
Gaussian Naive Bayes assumes that the likelihood of the features is Gaussian and makes its prediction based on the naive independence assumption.
Support vector machines try to separate the training data points using hyperplanes in a high-dimensional space. They try to find the hyperplanes which have the largest distance to the nearest data point of any feature.
Logistic regression models for the combinations of input variables the probabilities for the possible outcomes using a logistic function. The standard solver *liblinear* trains separate binary classifiers for each possible outcome in a one vs the rest fashion.

But first we will apply some preprocessing algorithms to the dataset. For scaling, we try min-max scaler which scales each feature to a value between 0 and 1 by default, standard scaler which removes the mean of each feature and scales it to unit variance and robust scaler which removes the median of each feature and scales it by default according to the interquartile range. We will try to remove some outliers as well, to test if it improves the performance. After the feature scaling and outlier removal, we will apply the feature transformation algorithm principal component analysis (PCA). Principal component analysis transforms the data into orthogonal components by calculating the variance. The highest variance in the data will be assigned to the first component. The second component will be the highest variance orthogonal to the first component and so on.
Feature transformation could emphasize the more important features. In the end, the model will be evaluated by using the accuracy score. The accuracy score will show the percentage of the correct guesses.

Benchmark

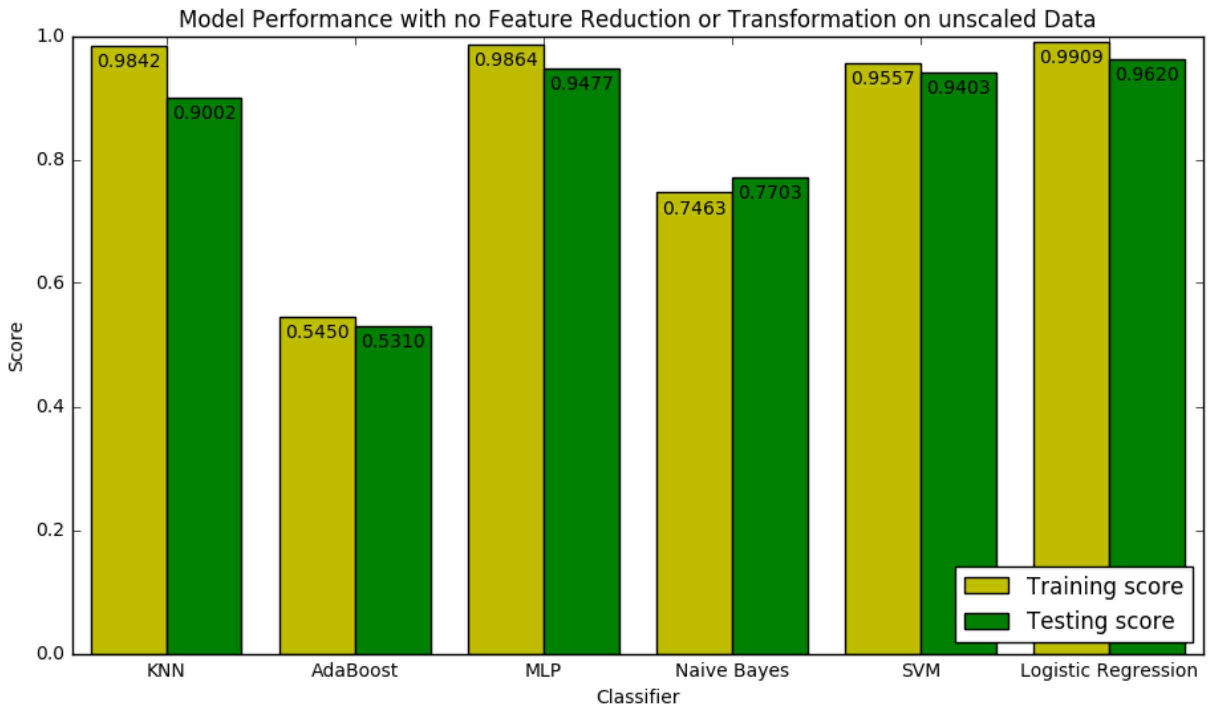
The dataset has already been used for some data analysis on kaggle. The user PedramNavid got a good accuracy of 0.9471 out of his project. I will take this accuracy as a benchmark model and will try to outperform it or at least to get a similar result.

Link to PedramNavid's project.

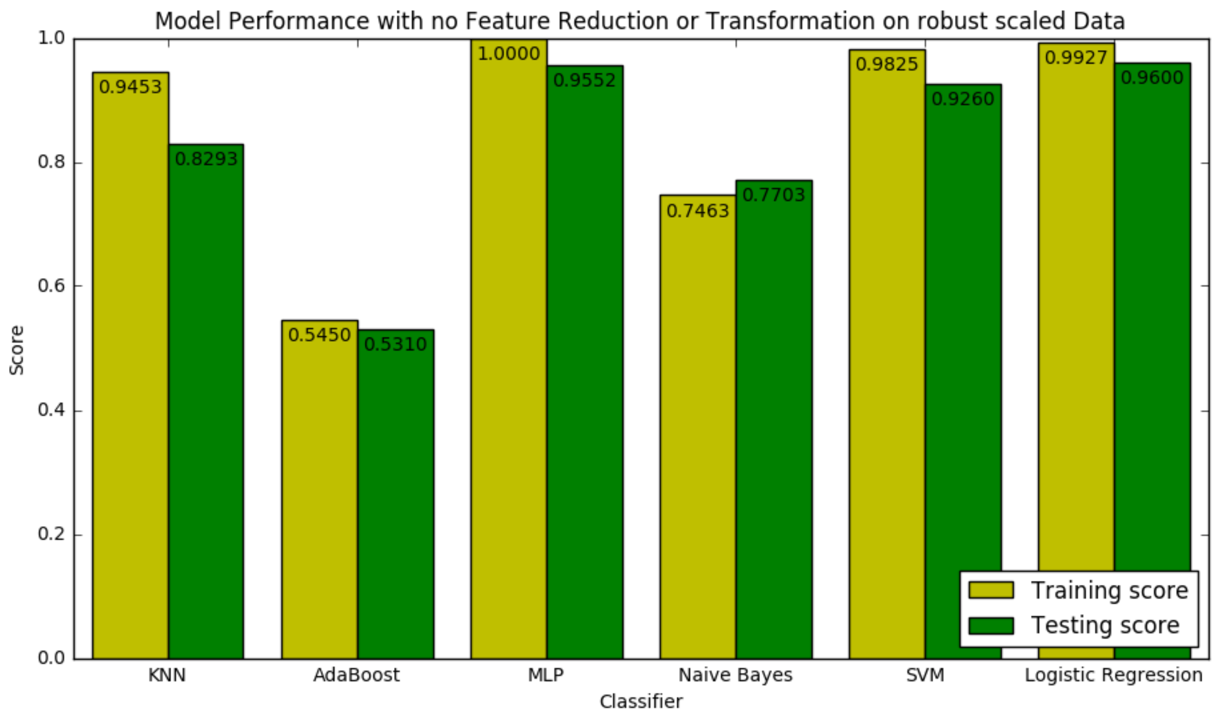
Methodology

Data Preprocessing

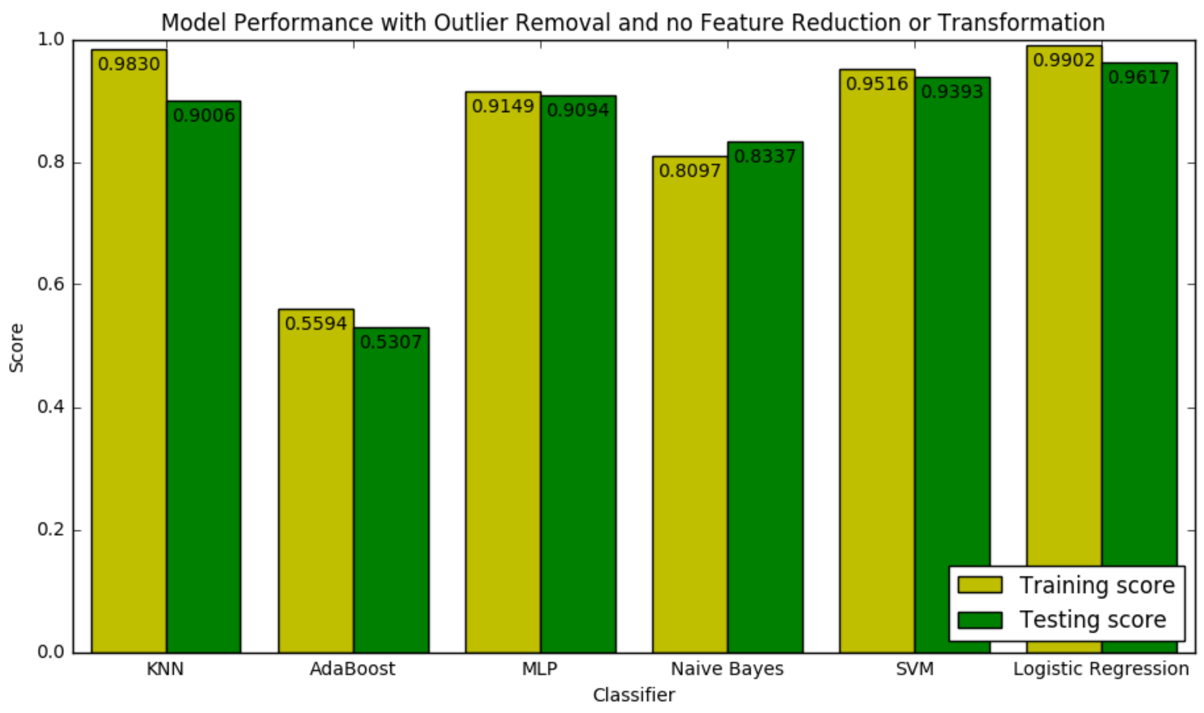
Before we start to preprocess the data, we apply the classifier to the raw dataset and plot the performance. This will create some benchmarks for us to guess whether the preprocessing algorithm will help to improve the performance of the final model. The following plot shows the results. The classifiers multi-layer perceptron, support vector machines and logistic regression look promising. AdaBoost, however, does not perform well at all on the dataset.



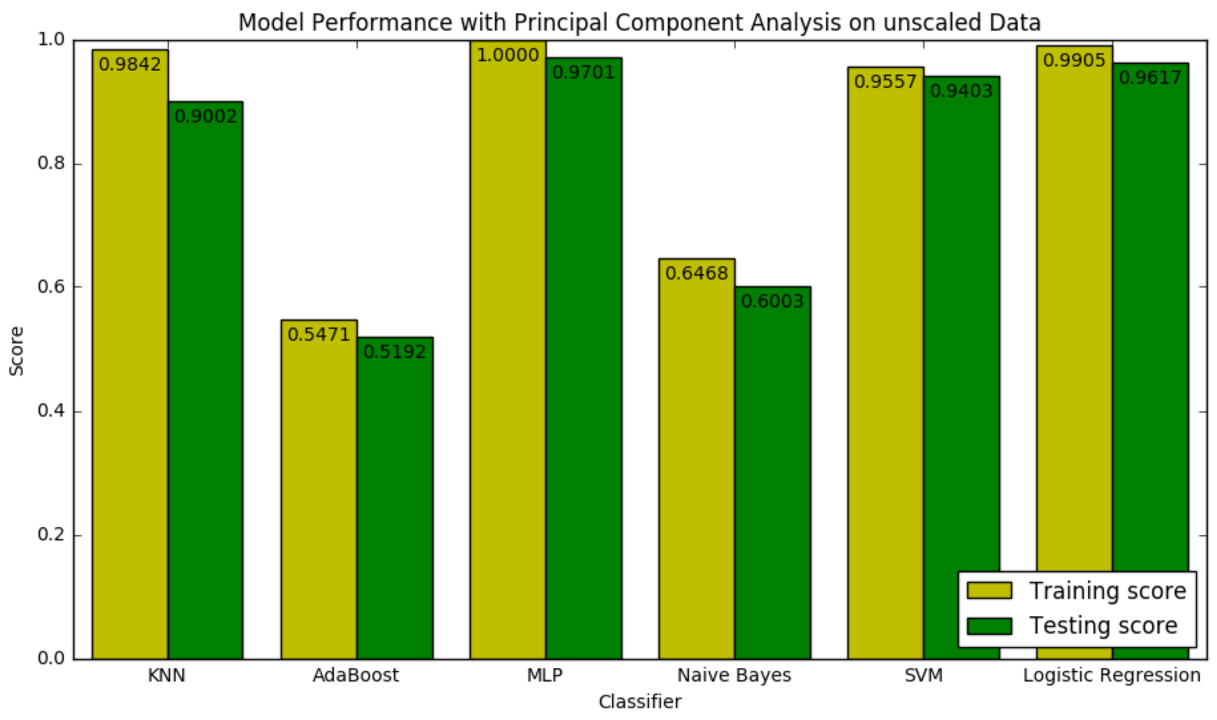
The best preprocessing algorithm for the classifiers seems to be the robust scaler. The robust scaler is a scaling algorithm which is very robust to outliers. It removes the median and scales the data according to its interquartile range if the default parameters are used. The following bar chart shows the performance after the robust scaler had been applied. The two best classifiers are still multi-layer perceptron and logistic regression. The accuracy score of the multi-layer perceptron was improved from 0.9477 to 0.9552. The accuracy score for the logistic regression, however, was decreased from 0.9620 to 0.9600.



Now we try to remove some outliers of the unscaled data and compare the performance of the classifiers to the performance without outlier removal. To identify outliers, we iterate through all the features and calculate the lower quartile and upper quartile of each feature. The difference between the upper quartile and the lower quartile is the interquartile range. To define a margin, the interquartile range will be multiplied by a factor which is usually around 1.5. The result of this calculation will be used to define a section of valid data. The section starts at the lower quartile minus the margin and ends at the upper quartile plus the margin. All data points outside this sections are considered as outliers and will be removed from the dataset. However, if we use the default factor 1.5 and remove all the data points which are considered as outliers for at least one feature, just 501 examples of the original 7,351 examples will be left. The reason is, the huge number of features in the dataset and the fact that they vary a lot. That is why we expand the margin a lot and choose a factor of 25 instead of 1.5. This huge margin leaves 6,804 examples of the original 7,351 examples in the dataset. The following plot shows the performance of the classifiers trained on these 6,804 examples. The performance of the multi-layer perceptron fell to 0.9094 and the performance of logistic regression was decreased to an accuracy score of 0.9617. This leads us to the conclusion that the outliers are important to the data and should not be removed.



The next step is feature transformation. After applying principal component analysis to the unscaled data as well as the robust scaled data, the principal component analysis applied to unscaled data turned out to be the combination with the best performance. The number of components were kept when principal component analysis was applied. Trying to reduce the number of components decreased the predictive power of the multi-layer perceptron model which got an amazing accuracy score of 0.9701 when all components were kept. The following plot shows the performance for each model.



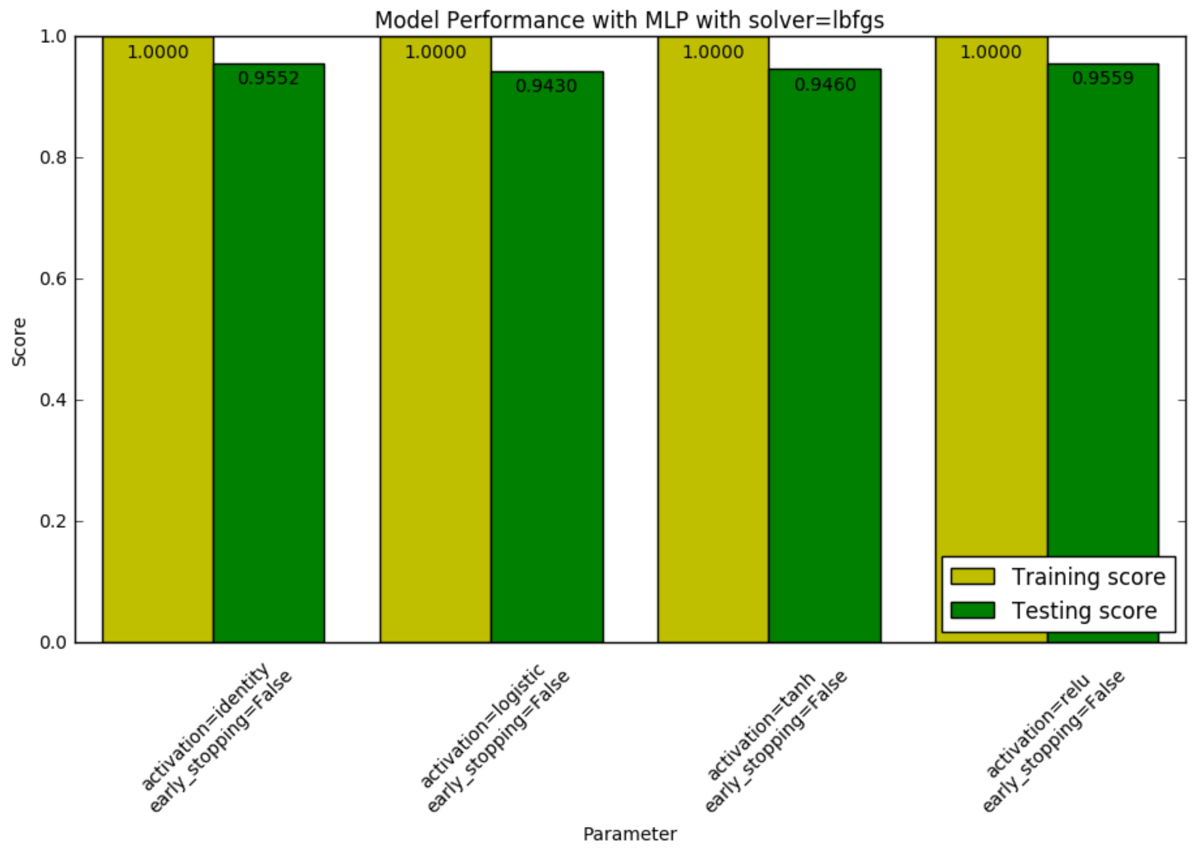
Implementation

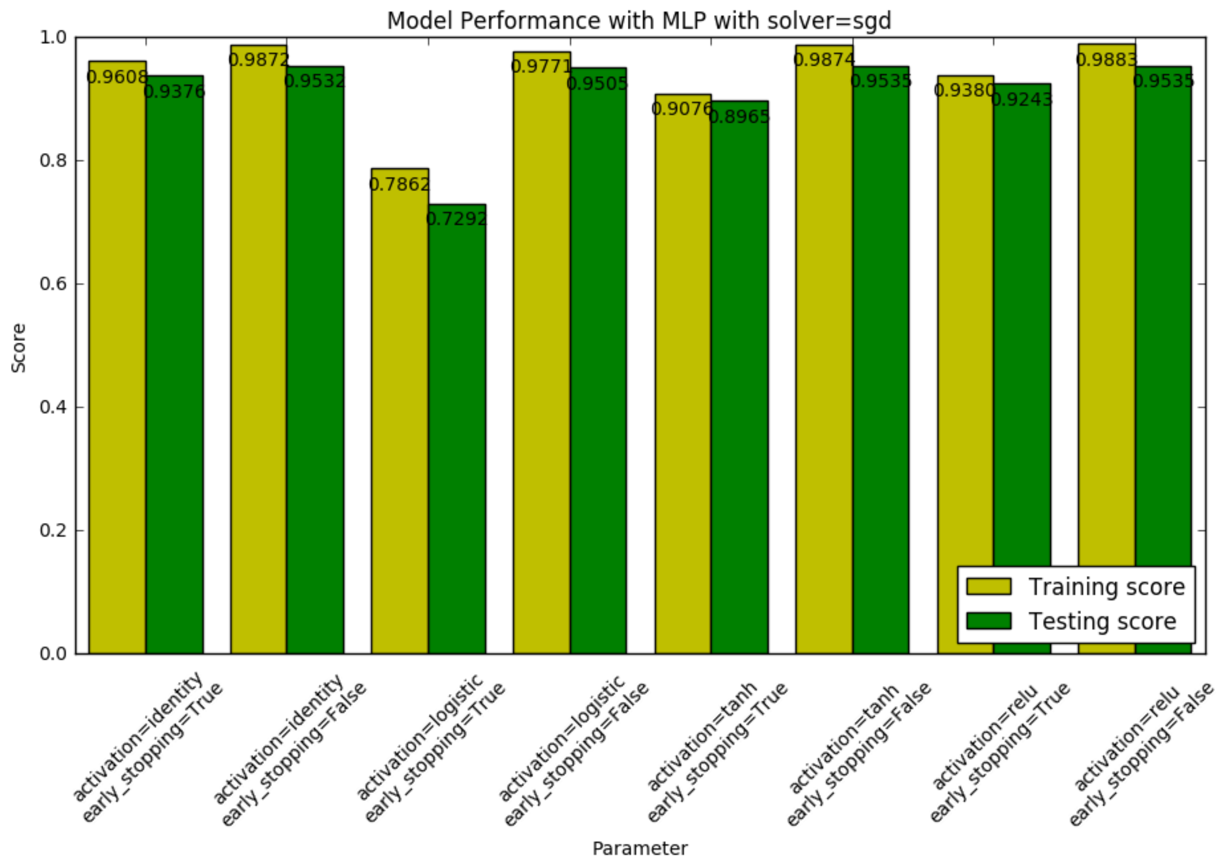
First the dataset has been imported, the activity labels extracted and the subjects dropped. Each scaler had been applied to the original data and its modified data was stored in a new variable. Feature transformation were applied to the scaled as well as the unscaled data. The hardest part to implement was the outlier removal. Because of the high number of features in the dataset, a lot of examples from the training data were considered to be an outlier for at least one feature. Since the outlier removal did not work out as good as hoped for, the outlier removal had not been exposed to any further feature transformation algorithm. Because several combinations of scaling, transformation and classification were tried on the data, we needed to define methods to run the classifiers and display the performance again and again with these different data variables. In the end, the application of the multi-layer perceptron on the principal component analysis transformed data turned out to perform best and got the highest accuracy score.

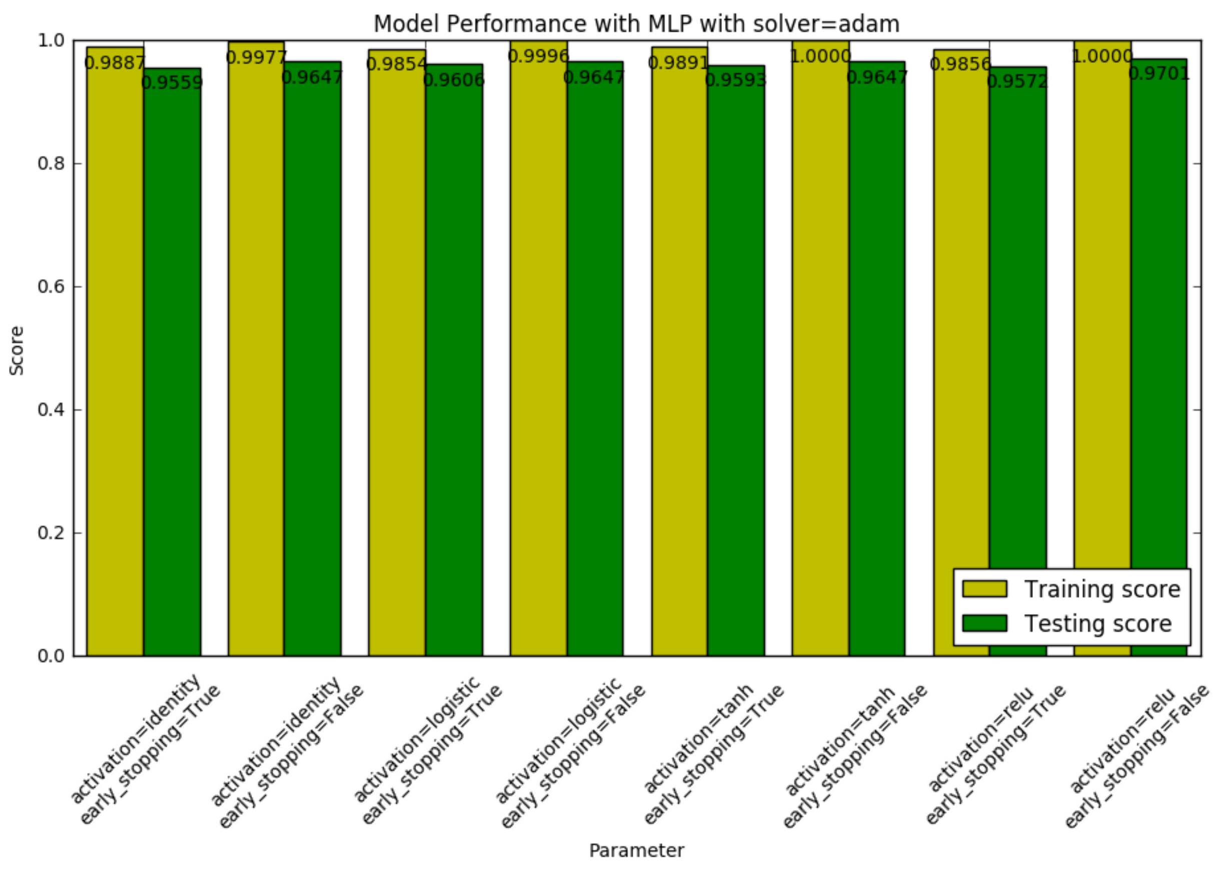
Refinement

The initial solution for the refinement is the result of transforming the features to components by principal component analysis and applying a multi-layer perceptron to this transformation. This initial solution got a great accuracy score of 0.9701, but we might still be able to tune the classifier. The tuning process will run the multi-layer perceptron with several parameter combinations on die PCA-transformed data. The accuracy score of each run will be displayed in bar charts to make them easily comparable.
We choose three parameters to tune the multi-layer perceptron. The first parameter is the *solver* which runs with *adam* by default. The other two options are *lbfgs* and *sgd*. The second parameter is the *activation* which runs with *relu* by default. Alternatives are *identity*, *logistic* and *tanh*. The third parameter which will be tuned is *early_stopping* which is *False* by default. The only other option is *True*, but the *early_stopping* parameter has no effect when the *solver lbfgs* is used. That is why *early_stopping=True* will just be tried with *sgd* and *adam*. The parameter *max_iter* will be set to *1,000* for the tuning scenario, because otherwise a convergence warning from the stochastic optimizer appears saying that the maximum iterations were reached and the optimization has not converged yet.
The following plots show the performance of each *solver* with all possible *activation* and *early_stopping* combinations. The *solver lbfgs* shows a constant performance for each *activation* parameter. But with an accuracy score between 0.9430 with *activation=logistic* and 0.9559 with *activation=relu*, it is not as high as the initial solution. The *solver sgd* is less constant. The accuracy scores with *early_stopping=True* are always lower than the accuracy scores with *early_stopping=False* and reach from 0.7282 with *activation=logistic* to 0.9376 with *activation=identity*. With *early_stopping=False* it reaches accuracy scores from 0.9505 with *activation=logistic* to 0.9535 with *activation=tanh* and *activation=relu*. These scores are also not as high as the initial solution. The default solver *adam* has by far the best and very constant performance. Again, all the runs with *early_stopping=True* got lower accuracy scores than *early_stopping=False* and reach from 0.9559 with *activation=identity* to 0.9606 with *activation=logistic*. The best accuracy score for the *solver adam* got the parameter *activation=relu* with 0.9701 which matches the initial solution. The other three parameters for *activation* got an equal accuracy score of 0.9647. Unfortunately, the tuning did not

improve the performance any further because the *solver adam*, the *activation relu* and *early_stopping=False* are the default values for the multi-layer perceptron.







Results

Model Evaluation and Validation

The final model is the result of a feature transformation using principal component analysis with all its default parameters and no component reduction, combined with a multi-layer perceptron classifier algorithm with all its default parameters. It was chosen as the final model, because it got the highest accuracy score with a value of 0.9701 which means it predicted 97.01% of the activities from the test examples correctly.
It is still required to validate the robustness of the model. A good way to do this is to look at the confusion matrix.

Predicted	**LAYING**	**SITTING**	**STANDING**	**WALKING**	**WALKING_DOWNSTAIRS**	**WALKING_UPSTAIRS**	**All**
True							
LAYING	536	0	1	0	0	0	537
SITTING	2	436	50	0	0	3	491
STANDING	0	12	519	1	0	0	532
WALKING	0	0	0	496	0	0	496
WALKING_DOWNSTAIRS	0	0	0	1	413	6	420
WALKING_UPSTAIRS	0	0	0	12	0	459	471
All	538	448	570	510	413	468	2947

It looks like the model does very well when it comes to the activities LAYING and WALKING. 536 of the 537 test examples where predicted correctly for the activity LAYING. For WALKING, all the 496 examples were correctly guessed. These are amazing recall scores for those two. The precision score for LAYING is also great, because 536 guesses of the 538 guesses were indeed LAYING activities. The only activity which has a higher precision score than LAYING is WALKING_DOWNSTAIRS. All the 413 guesses of WALKING_DOWNSTAIRS are correct. The weakness of the model is telling the difference between SITTING and STANDING. 50 of the 491 SITTING examples were guessed as STANDING, which means the recall score for SITTING must be low. Vice versa, 12 of the 532 examples for STANDING were misinterpreted as SITTING which is the third worst value in the matrix. The second worst value has WALKING_UPSTAIRS which was misinterpret as WALKING 12 times but has just 471 examples.

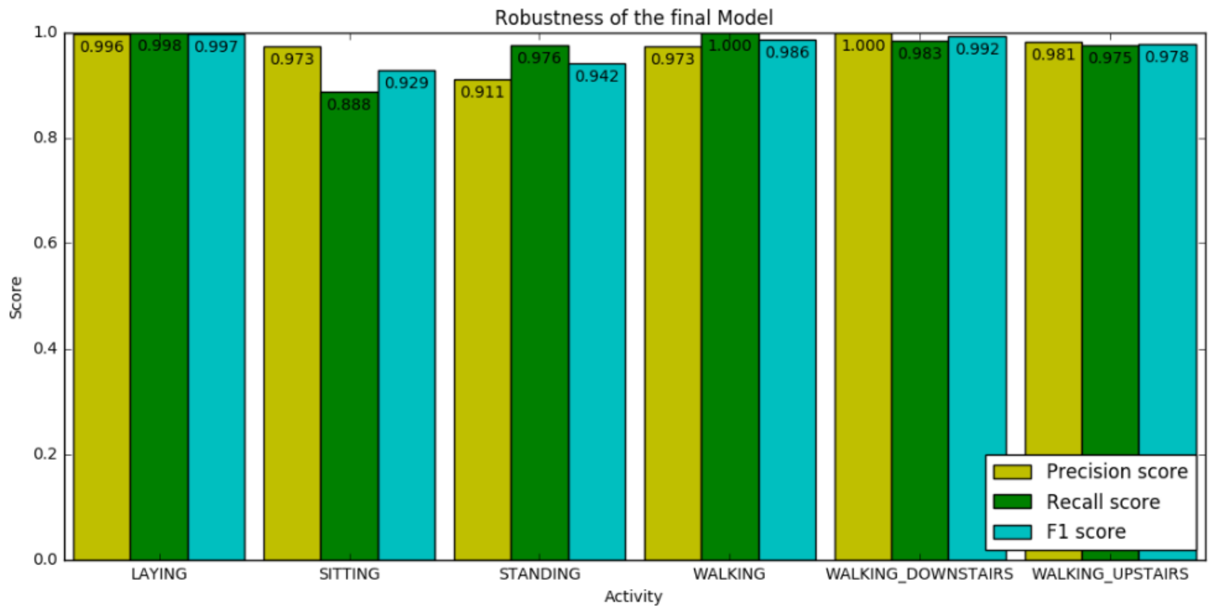
Justification

At the beginning of the project we set an accuracy score of 0.9471 as a benchmark, which was obtained in a project on kaggle from the user PedramNavid. Using principal component analysis and multi-layer perceptron, a higher accuracy score of 0.9701 has been reached. Comparing these two scores, I believe the new final model adequately solved the problem.

Conclusion

Free-Form Visualization

The following bar chart shows the precision score, recall score and the f1 score for each activity. The f1 score is a combination of the recall score and the precision score. It is a good method to show the robustness of a model.



LAYING has the highest f1 score of 0.997. On second place comes WALKING_DOWNSTAIRS with a f1 score of 0.992 followed by WALKING with a f1 score of 0.986 and WALKING_DOWNSTAIRS with a f1 score of 0.978. The f1 score shows the weakness of the model, too. STANDING got a f1 score of 0.942 and SITTING got a f1 score of 0.0929. They are confused more often than the other activities.

Reflection

The project's dataset has been obtained from kaggle together with a short overview. The task was to recognize the activity of humans while they are using a smartphone. The first step was to load the dataset into a pandas data frame and to display its first rows and columns to see how the data looks like. Some information like the data type, minimum values, maximum values, standard deviation or mean had been calculated to get a first impression. The number of examples for each activity had been counted to see if there is enough data for each activity to train and test on. The activity labels where extracted and stored separately. The subjects had been dropped from the dataset.
The first obstacle was to get an overview over all features since there are so many. It is not possible to show the correlation between all features within one correlation matrix. That is why three correlation matrixes of three sections of the data were displayed. One from the first features, one from some features further in the middle and one from the last features. This provided a deeper impression about the dataset and showed how variegated the features are.
After the data exploration, the min-max scaler, the standard scaler and the robust scaler had been used to scale the features and each of the resulting data had been applied to the six classifiers k-nearest neighbors, AdaBoost, multi-layer perceptron, Gaussian naive Bayes, support vector machines and logistic regression. The robust scaled data got promising accuracy scores which were about as promising as the accuracy scores of the unscaled data. That is why both were kept for feature transformation. Outlier removal has also been tried by defining a valid data section for each feature and removing the data points which did not lie within this section. But since the data varies in many ways, it was hard to find an algorithm which reliably finds outliers that should be removed (if there are any). The data, which resulted from the outlier removal attempt, was applied to the six classifiers. But they not perform as good as the data before the outlier removal. The outlier cleaned data had been dropped for the rest of the project.
The data transformation has been done on the robust scaled and unscaled data using principal component analysis. All components were kept in this process. Both, the scaled-transformed and the unscaled-transformed data, had been applied to the six classifiers and the accuracy scores have been calculated. The highest accuracy score got the unscaled-transformed data in combination with the multi-layer perceptron.
This model has been tuned with all available values for three parameters (solver, activation and early_stopping). But no combination improved the performance of the model, anymore. The final solution in this project is a model which had been feature transformed using principal component analysis with all default parameters and classified using multi-layer perceptron with all default parameters. It reaches an accuracy score of 0.9701. The robustness was verified by using the precision score, the recall score and the f1 score. All results have been visualized in plots. Finally, a comparison to the benchmark model had been made.

The final model performs beyond my expectations. To correctly guess about 97% of the activities is a good percentage. I guess the model could also be used in a general setting if the data is similar.

Improvement

I can imagine, that discovering and removing some outliers is still possible. It just needs the right idea and the right algorithm which can be applied to the dataset or some of its features. The high number of features make the removal difficult but not impossible.
Another improvement could be feature reduction. Maybe a closer look at the features and the correlation between them might help to remove some.
Additionally, some further parameters could be tuned for the classifier. Maybe, some work out well and can be changed for the final model.
Finally, another classifier can be tried out and applied to the dataset. I am sure a better solution exists or will exist one day. It always does. One must just find it!