# K-Nearest-Neighbors

The K-Nearest-Neighbors classification algorithm classifies the given data based the neighborhood of each data point. The neighborhood is determined by the distance to other data points. The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point and predict the label from these. The number of samples can be a user-defined constant (k-nearest neighbor learning) or vary based on the local density of points (radius-based neighbor learning). The distance can, in general, be any metric measure: standard Euclidean distance is the most common choice. Neighbors-based methods are known as non-generalizing machine learning methods, since they simply “remember” all of its training data. (Pedregosa, 2011)  
After pre-processing of the dataset, we selected the features according to the k highest scores using sklearn.feature\_selection.SelectKBest module. After trying with different values of ‘K’, I decided to go with k=10, for which we got better results. Split the data into training set (75%), and test set (25%). Training set will be used to fit the model, and test set will be to evaluate the best model to get an estimation of generalization error. Instead of having validation set to tune hyperparameters and evaluate different models, we’ll use 10-folds cross validation because it’s more reliable estimate of generalization error. This procedure was performed after the feature generation with featuretools library, polynomial feature generation and the PCA feature decomposition.

## Results

The results presented as table after different approaches. Measurement of success is F1 score and accuracy.

The F1 score is calculated with the following method: , where precision is the share of true positive results in all positive predictions and recall the share of predicted true positives of all positive results. All results are taken after a full GridSearch, with 10 folds, to find the best estimator on each dataset. Furthermore, a train-test split von 25% is used.

As a result of the unbalanced data (8:2) every accuracy score below 0.8 is considered as below guessing and not useful to predict the outcome precisely.

|  |  |  |
| --- | --- | --- |
| Used dataset | F1 score | Accuracy |
| Basic dataset, without NaN values | 0.35 | 0.74 |
| Basic dataset, without outliers | 0.32 | 0.74 |
| PCA decomposition dataset | 0.99 | 0.996 |
| Dataset with new features, generated with “featuretools” | 0.01 | 0.84 |
| Dataset with new polynomial features | 0.03 | 0.89 |

# Decision Trees

Decision Trees (DTs) are algorithms which classify each data point

by its features to build a tree structure to walk through. The nodes represent a

decision, e.g. “Credit Score > 500” and a junction True or False. The algorithm builds a

whole tree with nodes from the same kind to classify the features and if fed with

new features, to predict the outcome. In this evaluation four types of decision

trees are used. DTs are a non-parametric supervised learning method used for

classification and regression. The goal is to create a model that predicts the value

of a target variable by learning simple decision rules inferred from the data features.

The deeper the tree, the more complex the decision rules and the fitter the model (Pedregosa, 2011)

After pre-processing of the dataset, we selected the features according to the k highest scores using sklearn.feature\_selection.SelectKBest module. After trying with different values of ‘K’, I decided to go with k=10, for which we got better results. Split the data into training set (75%), and test set (25%). Training set will be used to fit the model, and test set will be to evaluate the best model to get an estimation of generalization error. Instead of having validation set to tune hyperparameters and evaluate different models, we’ll use 10-folds cross validation because it’s more reliable estimate of generalization error. This procedure was performed after the feature generation with featuretools library, polynomial feature generation and the PCA feature decomposition.

## Results

The results presented as table after different approaches. Measurement of success is F1 score and accuracy.

The F1 score is calculated with the following method: , where precision is the share of true positive results in all positive predictions and recall the share of predicted true positives of all positive results. All results are taken after a full GridSearch, with 10 folds, to find the best estimator on each dataset. Furthermore, a train-test split von 25% is used.

As a result of the unbalanced data (8:2) every accuracy score below 0.8 is considered as below guessing and not useful to predict the outcome precisely.

|  |  |  |
| --- | --- | --- |
| Used dataset | F1 score | Accuracy |
| Basic dataset, without NaN values | 0.31 | 0.73 |
| Basic dataset, without outliers | 0.32 | 0.74 |
| PCA decomposition dataset | 1.0 | 1.0 |
| Dataset with new features, generated with “featuretools” | 0.0 | 0.84 |
| Dataset with new polynomial features | 0.003 | 0.84 |

# References

Pedregosa, F. a. (2011). Scikit-learn: Machine Learning in {P}ython. *Journal of Machine Learning Research*, pp. 2825--2830.