

# MIS-64037- Advanced Data Mining and Analytics

(Group 2)

**Instructor: Dr. Rouzbeh Razavi**

Team Member	Involvement
Nicholas Golina	Model Building, Project Report
Devesh Petwal	Exploratory Analysis, Insights & Presentation
Harika Penjerla	Model Building, Project Report
Alfred Rajan	Model Building, Project Report

## Objectives

Machine learning plays an integral part in various aspects of the financial system. The team has been assigned the task to run the Underwriting department of a bank. This task involves not only identifying the potential customers that can be loaned money by the bank, but also determining the set of customers with the highest net return from the money that is loaned. The ultimate aim is to minimize risk by anticipating the customers that are likely to default and the potential severity of the losses. The process to achieve this will be through a combination of Classification and Regression modelling, while defining a loan threshold based on the probability of default and loss given default.

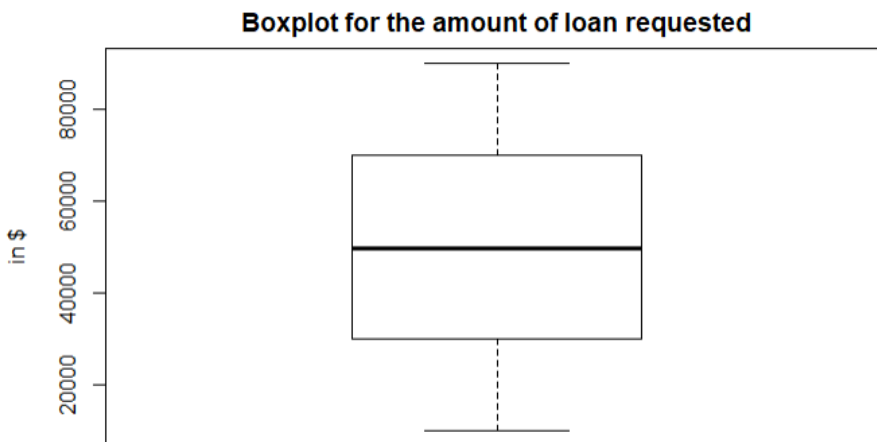
## Overview of Data

Here is an overview along with results from exploration and analysis of the data we had:

### Scenario 1 & 2:

- Total number of customers: 25,471
- Total loan requested: \$126,977,6911
- Maximum loan amount requested: \$89,995
- Minimum loan amount requested: \$10,002
- Proposed interest rate: 4.32%
- Missing values in the training data: 593,281

```
boxplot(test2$requested_loan, ylab = "in $", main = "Boxplot for the amount of loan requested", color = "red")
```

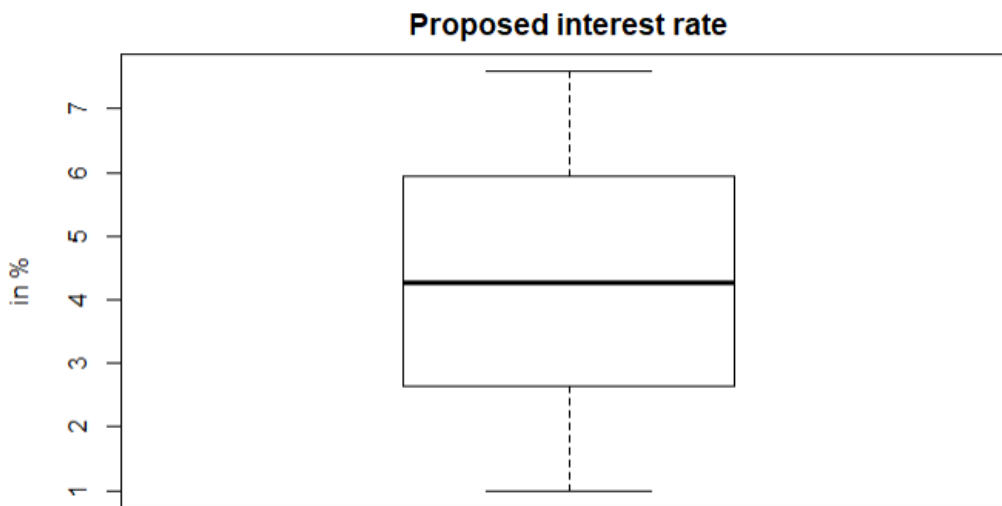


### Scenario 3:

- Total number of customers: 25,471
- Total loan requested: \$1,280,654,185
- Maximum loan amount requested: \$89,993

- Minimum loan amount requested: \$10,003
- Average proposed interest rate: 4.292058%
- Lowest proposed interest rate: 1%
- Highest proposed interest rate: 7.6%
- Missing values in the training data: 593,281

```
boxplot(test3$Proposed_Intrest_Rate, ylab = "in %", main = "Proposed interest rate", color = "red")
```



## Modeling Strategy

### Dimensionality Reduction Using Principal Component Analysis:

The data which has a large number of features are said to have high dimensionality. High Dimensionality increases the complexity and most importantly may result in overfitting of the model. In order to deal with the dimensionality, techniques were used to reduce the number of features by choosing the most important ones that still represent the entire data. All features with a standard deviation of .3 and below were eliminated as well as duplicate features. Additionally, the median was used to impute for missing values in the dataset. There are also many techniques like missing value ratio, backward feature elimination, forward feature selection, principal component analysis etc. In this project analysis Principal Component Analysis was chosen as one of the techniques for dimensionality reduction for classification model.

PCA is an unsupervised technique used to preprocess and reduce the dimensionality while preserving the original structure so that machine learning models can still learn from them and be used to make accurate predictions. The main idea of PCA is to reduce the dimensionality of data consisting of many variables correlated with each other, while

retaining variation present in the data up to maximum extent. Transforming the variables to a new set of variables called Principal Components which are orthogonal. The PC are the eigenvectors of a covariance matrix and hence they are orthogonal.

Importantly, the data on which PCA is used needs to be scaled and results are sensitive to relative scaling. The algorithm uses the concepts of the variance matrix, covariance matrix, eigenvector, eigenvalues pairs to perform PCA, providing a set of eigenvectors and its respective eigenvalues as its results. Eigen vectors represent the new set of axes of PC and eigenvalues represent the quantity of variance of that eigenvectors have. So, eigenvectors with more variance are required for further analysis of machine learning modeling. A new matrix is then constructed with K-eigenvectors, thereby reducing the n-dimensional data to k dimensions.

### Implementation of PCA in Python:

Step1: Standardize the data (Scaling)

Using “StandardScaler” to help standardize the dataset with mean=0 and variance =1 from package “sklearn.preprocessing”.

```
#Scale the Data
scaler_train = StandardScaler()
scaler_test_1_2 = StandardScaler()
scaler_test_3 = StandardScaler()
train_regression[train_regression_list]=scaler_train.fit_transform(train_regression[train_regression_list])
test_1_2[test_1_list]=scaler_test_1_2.fit_transform(test_1_2[test_1_list])
test_3[test_3_list]=scaler_test_3.fit_transform(test_3[test_3_list])
```

Step2: PCA model

Using the PCA function from package “sklearn.decomposition”.

```
[ ] from sklearn.decomposition import PCA
    pca=PCA(.90)
    pca.fit(x_train)
    x_train = pca.transform(x_train)
    x_test = pca.transform(x_test)
```

0.90 was chosen for the component's parameter. It means that scikit-learn chooses a minimum number of principal components with 90% of variance retained.

## Results of PCA Implementation:

Output of train dataset reduced to 94 variables.

x\_train.head()

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
0	8.231470	0.057976	3.875687	10.717900	2.455455	-4.103636	4.235567	-1.909154	1.053601	-0.741173	-0.555550	-0.710174	-1.610616	-0.475959	-2.114423	2.25
1	10.933857	1.618337	-4.192629	-4.972424	2.657922	0.837063	0.058746	0.264592	0.073743	-0.297202	-2.564860	-0.345889	-2.046360	-0.709252	-2.256470	1.90
2	20.140556	6.074444	-4.448961	-1.367447	4.017553	3.126987	0.120043	-3.502438	0.621121	-0.962926	-2.440934	0.552162	1.313802	0.716272	-0.063082	-0.78
3	3.194103	-18.353462	9.858296	5.825235	5.970544	2.391821	-5.115439	-3.364240	-2.839986	1.303598	0.125439	-0.867716	0.188776	0.609429	3.962599	4.56
4	-8.640175	-0.280851	-4.659150	-4.460708	-5.447700	0.074563	-4.018021	-0.154485	-1.345847	2.647152	0.839183	0.062455	-1.851433	-2.150126	-0.520857	-0.00

5 rows x 94 columns

## Regression and Classification:

In this project the following equations were used for expected returns:

Expected Loss:  $PD * Loan\_Value * LGD$

Expected Gain:  $Loan\_Value * Interest\_Rate * (1 - PD)$

Where PD is probability of default, a classification model was used for this.

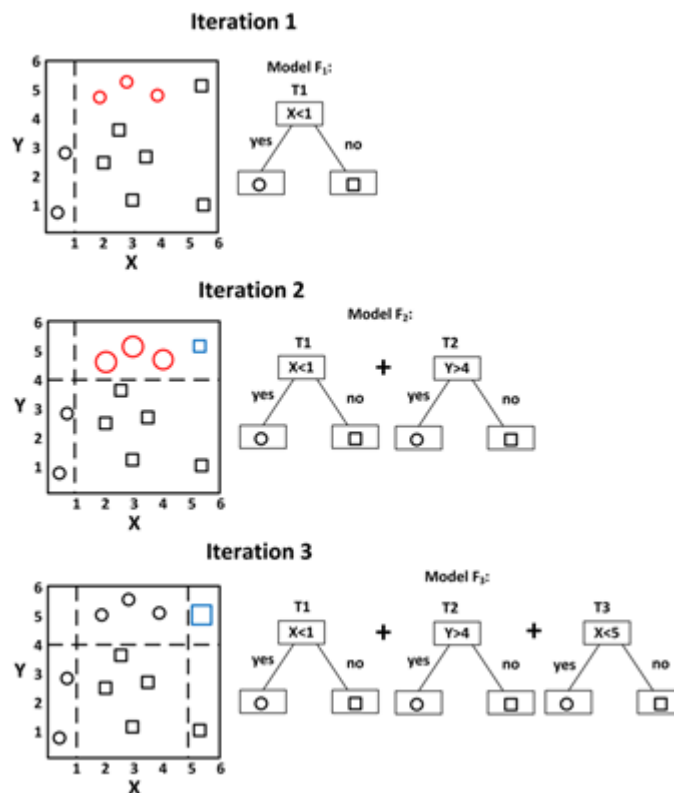
LGD is loss given default, a regression model was used for this.

For regression and classification models XGBoost was the choice of model. This is because the XGBoost model performed better than lasso regression and other choices of tree based models for classification.

## XGBoost:

Ensemble models in machine learning combine the decisions from multiple models to improve overall performance. Gradient Boosting is an ensemble learner which creates a final model based on collection of individual models. Boosting builds models from individual weak learners and sequentially putting more weights on instances with wrong predictions and high errors. When each ensemble was trained on a subset of the training set, this helps to improve the generalizability of model compared to a traditional gradient boosting model.

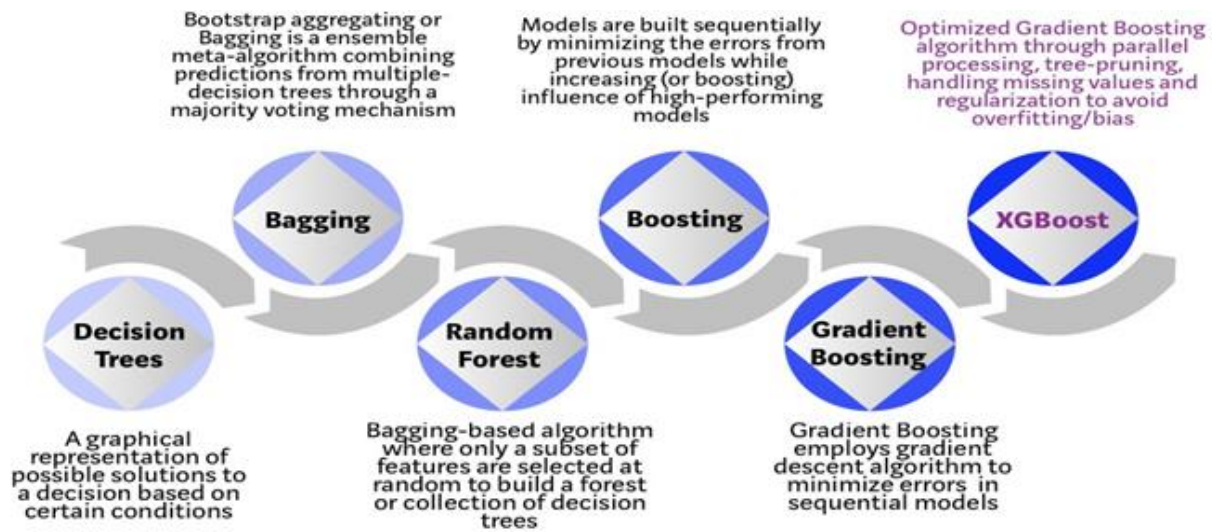
A simple illustration of gradient boosting.



XGBoost stands for eXtreme Gradient Boosting. It is a decision tree based ensemble machine learning algorithm under gradient boosting framework. It can be applied to both regression and classification problems. It is an optimized distributed gradient boosting library designed to be highly efficient, flexible, parallelization and portable.

Algorithm improvements over general gradient boosting are

1. Regularization: It penalizes more complex models through L-I and L-II to prevent overfitting.
2. Sparsity Awareness
3. Built-in Cross-Validation method.



## Implementation in Python:

Loading the appropriate library “xgboost”

Regression Model to find Loss given default

```
[ ] xgboost = xgb.XGBRegressor(objective='reg:squarederror', max_depth=6, random_state=12, learning_rate=.1,
                               n_estimators=100, nfold=4)
xgboost.fit(x_train, y_train)
predict_train = xgboost.predict(x_train)
predict_test = xgboost.predict(x_test)
```

Classification model to find probability of default

```
[ ] xgboost_class = xgb.XGBClassifier(random_state=12, learning_rate=0.05,
                                      n_estimators=100, max_depth=7, nfold=5)
xgboost_class.fit(x_train, y_train)
predict_train = xgboost_class.predict(x_train)
predict_test = xgboost_class.predict(x_test)
proba_train = np.array([a[1] for a in xgboost_class.predict_proba(x_train)])
proba_test = np.array([a[1] for a in xgboost_class.predict_proba(x_test)])
```

## Estimation of Model's Performance

While there are different methods to estimate a model's performance, in this project the team decided to go with certain performance estimators for the Classification model and the Regression model as follows:

Classification	Regression
<i>Accuracy, AUC, Confusion Matrix</i>	<i>Correlation Coefficient, Root Mean Squared Error</i>

### Classification Model:

The team decided to go with the ensemble modelling technique XGBoost Classifier to fit into the PCA transformed data. The model's performance metrics on two different scenarios are as follows:

- a) With PCA transformed data
- b) Without PCA transformed data.

Find below some of the model performance metrics for both the cases.

```
[ ] #Metrics for the Model
print('Accuracy of the model on training set: {:.2f}'.format(accuracy_score(y_train, predict_train)))
print('Accuracy of the model on test set: {:.2f}'.format(accuracy_score(y_test, predict_test)))
print('AUC of the model on training set: {:.2f}'.format(roc_auc_score(y_train, proba_train)))
print('AUC of the model on test set: {:.2f}'.format(roc_auc_score(y_test, proba_test)))
print('Classification Report for Training Set')
print(classification_report(y_train, predict_train))
print('Classification Report for Test Set')
print(classification_report(y_test, predict_test))
```

	Training		Test	
	Accuracy	AUC	Accuracy	AUC
<b>With PCA</b>	0.84	0.92	0.70	0.62
<b>Without PCA</b>	0.95	0.98	0.91	0.69



```

Accuracy of the model on training set: 0.84
Accuracy of the model on test set: 0.70
AUC of the model on training set: 0.92
AUC of the model on test set: 0.62
Classification Report for Training Set

```

	precision	recall	f1-score	support
0.0	0.89	0.77	0.83	61672
1.0	0.80	0.90	0.85	61672
accuracy			0.84	123344
macro avg	0.84	0.84	0.84	123344
weighted avg	0.84	0.84	0.84	123344

```

Classification Report for Test Set

```

	precision	recall	f1-score	support
0.0	0.93	0.73	0.82	10949
1.0	0.13	0.41	0.20	1051
accuracy			0.70	12000
macro avg	0.53	0.57	0.51	12000
weighted avg	0.86	0.70	0.76	12000

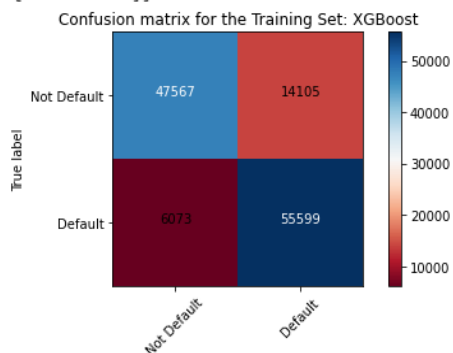
The team also used the Confusion Matrix to compare the model results with the ground results.

### With PCA:

```
[ ] cm_train = confusion_matrix(y_train, predict_train)
plot_confusion_matrix(cm_train, ['Not Default', 'Default'], normalize=False, title='Confusion matrix for the Training Set: XGBoost')
```




Confusion matrix, without normalization  
[[47567 14105]  
[ 6073 55599]]

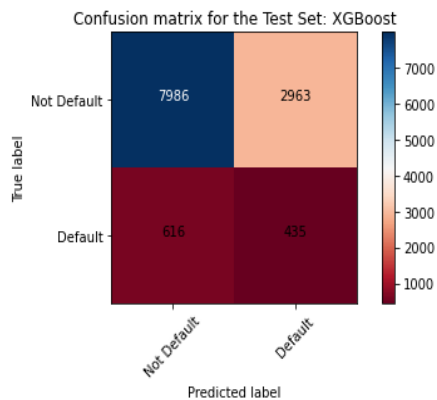


A breakdown of the above Confusion Matrix would be as follows:

True Positive	55599
False Positive	14105
True Negative	47567
False Negative	6073

```
[ ] cm_test= confusion_matrix(y_test, predict_test)
    plot_confusion_matrix(cm_test, ['Not Default', 'Default'], normalize=False, title='Confusion matrix for the Test Set: XGBoost')
```

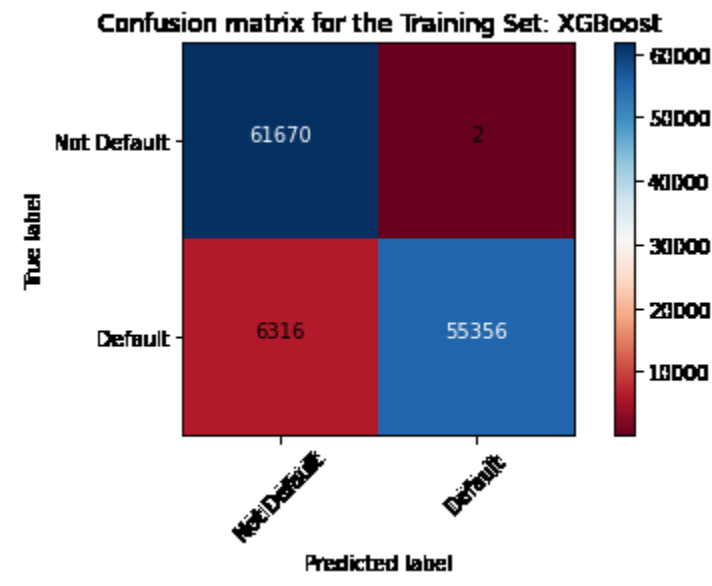
 Confusion matrix, without normalization  
[[7986 2963]  
 [ 616 435]]



A breakdown of the above Confusion Matrix would be as follows:

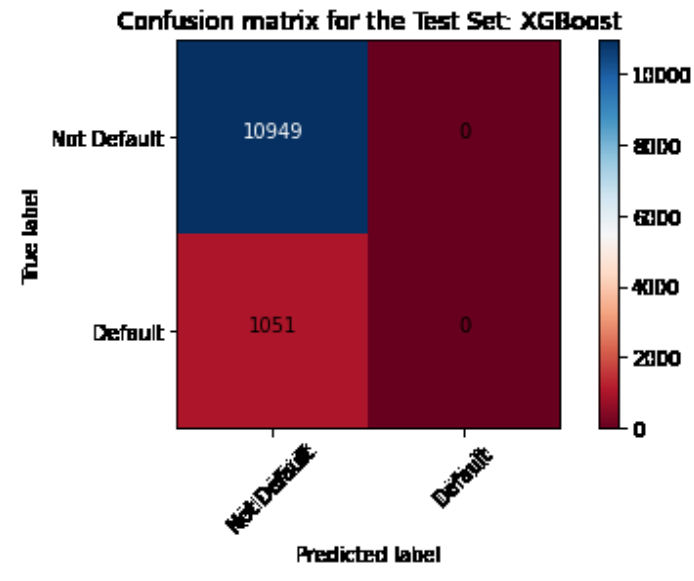
True Positive	435
False Positive	2963
True Negative	7986
False Negative	616

Without PCA:



A breakdown of the above Confusion Matrix would be as follows:

True Positive	55356
False Positive	2
True Negative	61670
False Negative	6316



A breakdown of the above Confusion Matrix would be as follows:

True Positive	0
False Positive	0
True Negative	10949
False Negative	1051

Though there are no considerable differences between both the models (PCA and non-PCA) in terms of AUC and Accuracy, the Confusion matrix draws a different picture. As seen above, the non-PCA model was not able to identify any positive cases for the test dataset which is a very undesirable outcome. Hence, the team decided to go with the PCA transformed data model.

### Regression Model:


As stated earlier, the ensemble modelling technique XGBoost was used to build regression models to predict the LGD (Loss Given Default) values pertaining to each candidate.

The model was built on both PCA transformed dataset and non-transformed dataset. The model's performance was estimated as follows:


	Training		Test	
	R^2	Root Mean Squared Error	R^2	Root Mean Squared Error
<b>With PCA</b>	0.8861471311659632	3.961810366517451	0.24276432879729526	9.928317492388537
<b>Without PCA</b>	0.9231895738022162	3.2541066494705726	0.39443747216177416	8.878500455766854

```
[ ] xgboost = xgb.XGBRegressor(objective='reg:squarederror', max_depth=6, random_state=12, learning_rate=.1,
                                n_estimators=100, nfold=4)
xgboost.fit(x_train, y_train)
predict_train = xgboost.predict(x_train)
predict_test = xgboost.predict(x_test)
```

```
[ ] print('Train Root Mean Squared Error: {}'.format(sqrt(mean_squared_error(y_train, predict_train))), 'Train R^2: {}'.format(r2_score(y_train, predict_train)))
```

 Train Root Mean Squared Error: 3.2541066494705726 Train R^2: 0.9231895738022162

```
[ ] print('Test Root Mean Squared Error: {}'.format(sqrt(mean_squared_error(y_test, predict_test))), 'Test R^2: {}'.format(r2_score(y_test, predict_test)))
```

 Test Root Mean Squared Error: 8.878500455766854 Test R^2: 0.39443747216177416

The values from the above table shows that the performance metrics are better for the XGBoost regression model built on non PCA dataset in comparison to the model built on PCA transformed dataset. R^2 values are higher, and the Root Mean Squared Error values are lower for the non PCA dataset. Hence, the team decided to go with the non PCA transformed data model.

## Insights and Conclusions

With the models chosen the results were as followed for scenario 1 and 2:

- Loan applications accepted: 3,352.
- Loan applications denied: 22,119.
- Amount of loan disbursed: \$ 164,442,425.
- 13.1% of total loan applications were accepted.
- Expected gain: \$ 577,543,556
- Expected loss: \$ 418,737,602
- Net return: \$ 158,805,953

For scenario 3:

- Loan applications accepted: 3,574.
- Loan applications denied: 21,897.
- Amount of loan disbursed: \$ 178,139,206.
- 16.3% of total loan applications were accepted.
- Expected gain: \$ 629,582,495
- Expected loss: \$ 466,037,645
- Net return: \$ 163,544,849

The takeaway from this is that a relatively conservative criteria of the risk of default, while a quasi conservative/moderate criteria of the potential loss from default is the optimal approach to maximizing returns from the underwriting process. Future models should use greater datasets, while providing greater visibility over the specific characteristics of features used.

## References:

- <https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d>
- [https://www.shirin-glander.de/2018/11/ml\\_basics\\_gbm/](https://www.shirin-glander.de/2018/11/ml_basics_gbm/)
- <https://towardsdatascience.com/dimensionality-reduction-does-pca-really-improve-classification-outcome-6e9ba21f0a32>
- <https://scikit-learn.org/stable/modules/preprocessing.html>
- <https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>
- <https://stackabuse.com/implementing-pca-in-python-with-scikit-learn/>
- <https://machinelearningmastery.com/estimate-performance-machine-learning-algorithms-weka/>