NATIONAL BANK OF KAZAKHSTAN

CONSUMER CREDIT RISK ANALYSIS VIA MACHINE LEARNING ALGORITMS

Monetary Policy Department
Working Paper №2021-4

Shalkar Baikulakov

Zanggar Belgibayev

The Working Paper and Analytical Note series of the National Bank of Kazakhstan (the "NBK") are designed to disseminate the results of the NBK's studies as well as other scientific and research works of the NBK staff. The results of research activity are disseminated with a view to encourage discussions. The views expressed are those of the authors and do not necessarily reflect the official views of the NBK.

Consumer credit risk analysis via machine learning algorithms

NBRK – WP – 2021 – 2

Consumer credit risk analysis via machine learning algorithms

Baikulakov Shalkar[1]
Belgibayev Zanggar[2]

Annotation

This project is an attempt to assess the creditworthiness of individuals through machine learning algorithms and based on regulatory data provided by second-tier banks to the central bank. The assessment of the creditworthiness of borrowers can allow the central bank to investigate the accuracy of issued loans by second-tier banks, and predict potential systematic risks. In this project, two linear and six nonlinear classification methods were developed (linear models – Logistic Regression, Stochastic Gradient Descent, and nonlinear - Neural Networks, kNN, Decision tree, Random forest, XGBoost, Naïve Bayes), and the algorithms were compared based on accuracy, precision, and several other metrics. The non-linear models illustrate more accurate predictions in comparison with the linear models. In particular, the non-linear models such as the Random Forest and kNN classifiers on oversampled data demonstrated promising outcomes.

The key words: *consumer credits, machine learning, bank regulation, stochastic gradient descent (linear model), logistic regression (linear model), kNN (neighbors), random forest classifier (ensemble), decision tree (tree), gaussian NB (naïve bayes), XGBoost, Neural network (MLP classifier)*

Classification JEL: G21, G28, E37, E51.

[1]Baikulakov Shalkar – Director of Project Management Office, Center for the Development of Payment and Financial Technologies.
E-mail: sh.baikulakov@payfintech.kz

[2]Zanggar Belgibayev – Chief analyst, Monetary Analysis Division, Monetary Policy Department, National Bank of Kazakhstan.
E-mail: zanggar.belgibayev@nationalbank.kz

Content

1. Introduction

The expansion of consumer loans is the main factor behind the growth of retail loans in recent years. An issuance of consumer loans may lead to economic growth in the country, but at the same time, it should be noted that excessive financing by banks can cause the emergence of systemic risks. The rapid growth of consumer lending can be explained by the emergence of accurate data and instruments to assess the credit risk of individuals as well as a stabilization of the economic situations and noted increase in the well-being of the population.

The rapid expansion of consumer lending carries a number of systemic risks, which are associated, first of all, with the growing level of debt burden on certain segments of the population. In case of a drop in the real income of the population, the banking system may deal with massive defaults on consumer loans. As a consequence, it can lead to a decline in aggregate demand in the economy. Such a development of events might significantly negatively affect the state of the economy.

Reviewing studies of other scholars on applying machine learning algorithms, we can conclude that the most popular method of analyzing the credit risks of consumer loans on big data set is the employment of non-linear models such as Neural Networks, kNN, Decision tree, Random forest, XGBoost, Naïve Bayes and models such as Logistic Regression, Stochastic Gradient Descent. It should also be noted that the majority of authors used data from credit bureaus and second-tier banks, however, this research was conducted based on regulatory data collected by the central bank. Consequently, there might be some discrepancies in the approaches that might give minor differences.

The first section is a review of the literature, in which similar works by other authors are reviewed. The second section describes the methodology of the study, as well as a list of predictors used. Next comes the discussion section of the results, in which the authors describe the results of the assessment. The findings of this study are the final section of the work.

2. Literature review

Grier (2012) stated that 5 parameters must be considered for consumer credit analysis. The first parameter is a character that refers to the applicant's reputation. Nowadays, credit managers have access to credit bureau report which shows the credit history of a consumer before deciding to issue a loan. The second factor is capacity which is related to the income and the existing financial liabilities of the consumer. Capital is the third parameter that indicates the down payment that the borrower can afford. The external factors such as the condition in the job market and general economic conditions are considered as the fourth factor. The last parameter is collateral.

Credit scoring techniques estimate the probability of repayment of consumers using the information in a credit bureau report and the credit application (Grier, 2012). It estimates the probability of repayment of customers. Application scoring is the first type of credit scoring model that evaluates the application of new consumers based on the parameters, such as capital, capacity, and so on. Another model such as the behavior scoring model assesses the repayment ability of consumers, warning about the potential delinquent accounts.

One of the goals of using the credit scoring technique is to decrease credit losses in the future. Besides traditional techniques, banks recently started to integrate machine learning (ML) algorithms in credit scoring. Henley and Hand (1996) compared machine learning techniques such as kth nearest neighbor (kNN) classification method with traditional credit scoring techniques such as linear, logistic regression, and decision tree. The algorithms were tested for bad debt risk and the technique with the lowest result was considered as the best method. In spite of the insensitivity of kNN classification to the parameters, it gave the best result. In addition, it takes a few seconds to assess the consumer and provide justifications for refusing to issue a loan (Henley and Hand, 1996).

Addo, Gueran, and Hassani (2018) applied logistic regression, random forest and gradient boosting classifier, and deep learning model for credit risk analysis of companies. Various data sets were used, and then 10 most important features were chosen and the same techniques were used to compare the results. The best algorithm is supposed to show the highest area under the curve (AUC) and the least root means square error (RMSE). Binary classifiers outperformed deep learning models, and the best performance belongs to the gradient boosting classifier.

Regression models of machine learning are also used to evaluate consumer credit applications. Munkhdalai et al. (2019) compared machine learning algorithms with human-expert-based models such as the FICO credit scoring system. Survey of Consumer Finances (SCF) data was used as a data set and various machine learning regression methods were applied to the data set. The result demonstrated that the credit losses would be lower if the lending institutions started to use machine learning from 2001. Additionally, deep neural networks and xgboost algorithms showed higher accuracy.

Brown and Mues (2012) tested the suitability and accuracy of classification techniques, such as logistic regression, neural network, decision tree, gradient

boosting, least-square support, and random forest, to imbalanced loan data set. The undersampling method was applied to the data set, and then the imbalance of the data set was increased progressively to assess machine learning classification techniques. The result indicates that random forest and gradient boosting classifiers dealt well with imbalanced data, while decision tree, quadratic discriminant analysis (QDA), and kNN classifier performed worse than other methods.

There are also several algorithms in machine learning besides the abovementioned models. Baesens et al. (2003) tested kernel-based support vector machine and least-squares support vector machine, and also other popular machine learning classifiers to real-life credit scoring data sets including Benelux and UK financial institutions' data sets. The result indicates that the neural network classifier and kernel-based support vector machines performed very well. The author also mentioned the good performance of linear discriminant analysis and logistic regression. 41 classifiers including novel credit scoring methods applied to the same data sets (Lesmann, Baesens, Seow, and Thomas, 2015). The result of the research shows that artificial neural networks (ANN) performed better than extreme learning methods (ELM), random forest (RF) better than rotation forest (RotFor). However, the methods cannot give insightful explanations of the model and future researches should be done to achieve this goal. Random Forest classifier was chosen as a benchmark as it has the capability to provide explanatory insights for fundamental analysis.

Tsai and Chen (2010) developed four hybrid machine learning methods to compare with simple classifiers. A hybrid algorithm is a combination of two machine learning methods. In this research, classification and clustering methods were chosen and four different methods were selected. The result indicates that the combination of logistic regression (LR) and neural networks showed the highest prediction, while 'clustering + clustering' was the least accurate algorithm.

In 2015, the participants implemented XGBoost in 17 out of 29 Kaggle challenge winning solutions. The algorithm was implemented for store sales prediction, web text classification, customer behavior prediction, malware classification (Chen & Guestrin, 2016). In this research, the algorithm will be used for credit analysis and compared with other methods.

In this project, two linear and 6 nonlinear classification methods were applied to the dataset that will be described in the next section. The algorithms were compared based on accuracy, precision, and several other metrics.

3. Research methodology and data

Machine learning algorithms were used for the analysis of the consumer credit portfolios of Kazakhstani banks. The data was obtained from the Credit Register provided by second-tier banks to the National Bank of Kazakhstan (NBK). The loan with missing and unreliable parameters was cleaned out. In addition, loans with late payments of no longer than 90 days were removed from the data. If the delinquency of loan payment is more than 90 days, the loan is declared as a non-performing loan (NPL). Performing loans has no payment delinquency from the consumer side.

Table 1

Data

Parameters	Type
Region	Nur-Sultan, Almaty and 15 regions
Currency	KZT, RUB, USD and EUR
Type	Personal loan and credit card
Reason	Consumer credits and auto loans
Gender	Male or female
Citizenship	Local or Foreigner
Interest Rate	Ranges from 0% to 56%
Credit amount	Ranges from 10 000 to 15 million KZT
Age	Ranges from 18 to 99 years

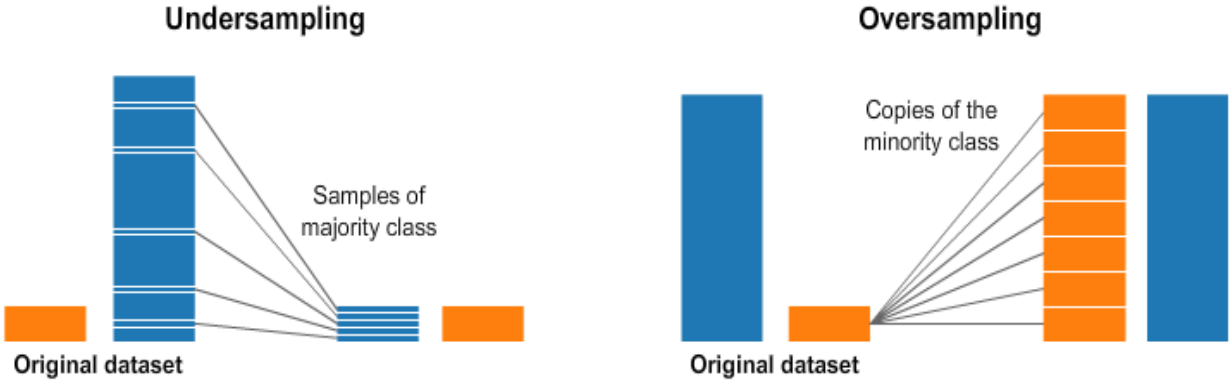
Source: National Bank of Kazakhstan

Resampling techniques for imbalanced data

In order to improve the accuracy of applied machine learning algorithms, resampling techniques such as undersampling and oversampling were utilized. Since the number of good loans was considerably more than the non-performing loans, the data was balanced and two techniques were compared.

Graph 1

Resampling strategies for imbalanced datasets



Source: Alencar, 2017

Random undersampler and oversampler are the most popular and simplest resampling strategies. The first strategy creates a new class with an equal size of minority class taken from the majority class, whereas the second strategy duplicates the minority class several times until the length of majority and minority classes will be equal. The drawback of a random undersampler is the loss of information. However, it is the only suitable undersampling strategy for a dataset that consists of categorical and continuous variables. Random oversampler leads to overfitting as it copies the minority class (Alencar, 2017). Therefore, another appropriate resampling method was selected.

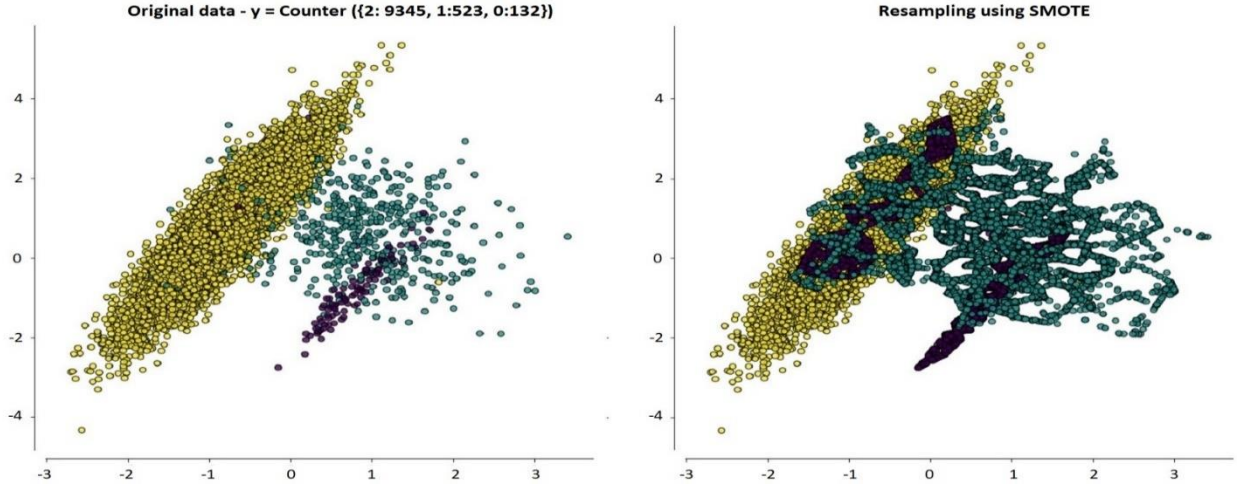
SMOTE (Synthetic Minority Oversampling Technique) is another popular oversampling method that draws new samples of minority classes using its existed data. It draws the lines between an example and the nearest 5 neighbors and creates

a new sample along that line. Graph 2 illustrates how new samples were created, thereby providing additional information to the machine learning model (Brownlee, 2020).

SMOTE cannot be applied to the above mentioned dataset as it does not handle categorical features. Therefore, Synthetic Minority Oversampling Technique-Nominal Continuous (SMOTE-NC) is selected as the oversampling method. It works as a SMOTE method for a continuous dataset, and the categorical variable of the new sample is the value of the majority of the k-nearest neighbors (Chawla, Bowyer, Hall and Kegelmeyer, 2002).

Graph 2

Oversampling minority classes using SMOTE



Source: Lemaitre, Nogueira, Oliveira and Aridas, n.d.

Preprocessing

Preprocessing is important task to make the data readable. Six parameters of dataset are categorical data and it has to be converted to numerical using encoder. Scikit-learn library (Python) provides techniques that convert discrete features to one-hot numerical array. It also has models which split the data into training and testing dataset (du Boisberranger, n.d.).

The next important part of preprocessing is feature scaling of the dataset. Machine learning algorithms commonly use Euclidean distance measure; thus it is sensitive to magnitudes of parameters. For instance, the algorithms neglect other parameters in the dataset because of large values of credit amount. Therefore, feature scaling is required to normalize the dataset. It also decreases the cost: credit scoring techniques analyze the normalized dataset faster. Abovementioned python library provides standard feature scaling techniques that normalize the dataset through the formula:

$$z = \frac{x-\mu}{\sigma}$$ (du Boisberranger et al., n.d.)

whereas, μ is mean and σ is standard deviation of the data.

Principal component analysis

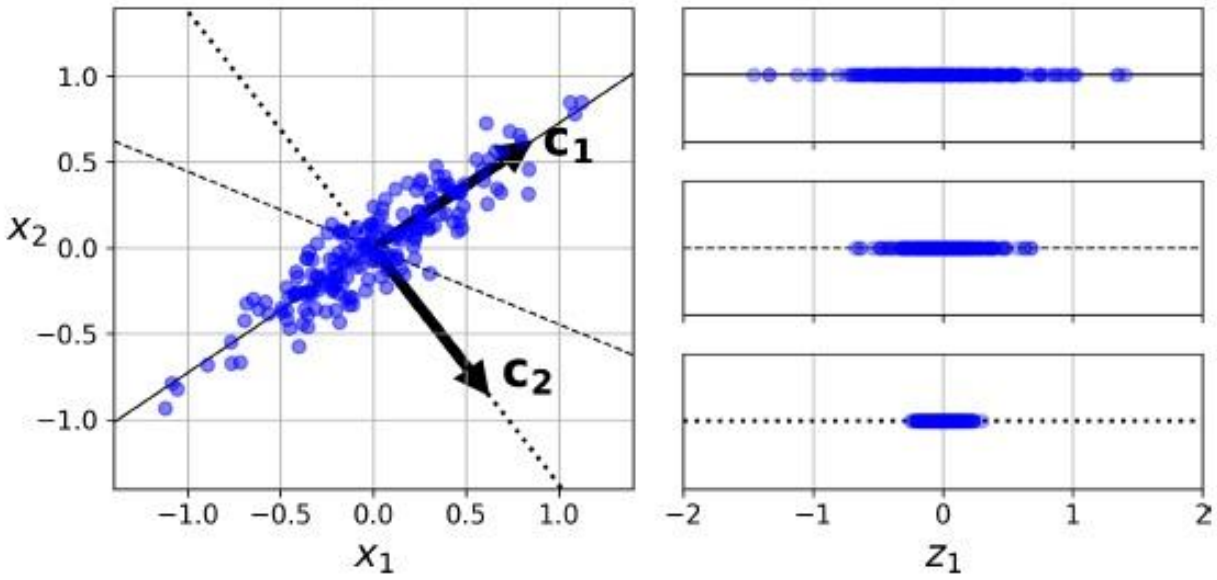
Principal component analysis (PCA) is one of the well-known dimensionality reduction techniques. It helps to decrease the dataset dimension with minimal loss

of information (Mueller & Guido, 2017). Consequently, it will decrease the computation time of an algorithm.

The algorithm selects the right hyperplane and projects the data onto that hyperplane. After the projection, the variance of the data is computed based on each new axis called the principal component. The first component preserves maximum variance, while the last component - least variance.

Graph 3

Principal component analysis

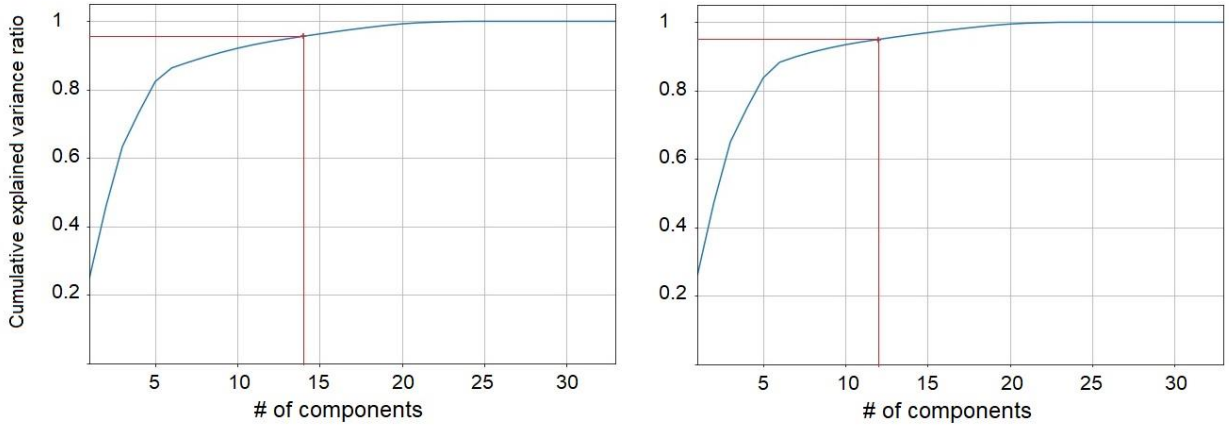


Source: Geron, 2017

Principal component analysis performed on both balanced datasets. The result of the undersampled dataset indicates that the first 14 components (columns) are enough to capture the 95% of the total variance of the dataset and the other 19 components can be erased. It is enough to use only 12 components of an oversampled dataset to capture 95% of the total variance.

Graph 4

Cumulative explained variance ratio of PCA



Source: compiled by the author

In this research, two linear and six nonlinear machine learning methods were applied to the dataset. The main goal to find out which algorithm outperforms others. In addition, the research helps to compare and contrast linear and nonlinear algorithms.

The feasibility of handling a large dataset is the main criteria for algorithm selection. Therefore, well-known algorithms such as linear and kernel support vector classifier (SVC) were not considered in the research. Logistic regression, stochastic gradient descent (SGD) classifier, Naïve Bayes classifier, kth nearest neighbors (kNN), decision tree classifier, random forest classifier, multilayer perceptron (MLP) classifier (neural network classifier), and extreme gradient boosting (XGB) classifier were implemented, and the results were discussed. Several hyperparameters of nonlinear algorithms will be neglected due to the inability to handle large datasets.

Logistic Regression

Despite its name, logistic regression is used for classification. The algorithm calculates the probability based on the training dataset according to the formula:

$$P(y = 1|x) = \frac{1}{1+e^{-(w_0+\boldsymbol{W}^T x)}}$$ (Baesens et al., 2003)

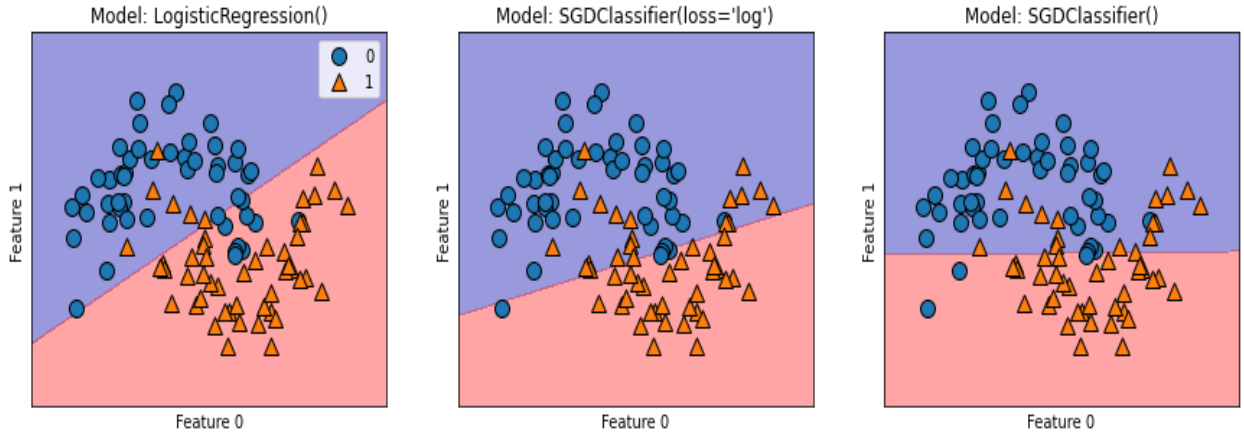
where x is input data, w_0 is a scalar intercept and W is a parameter vector. If the probability is more than 50%, the input data is classified as positive.

The important hyperparameter such as solver and regularization parameters were tuned to increase the accuracy. 'Liblinear' solver is generally used for a small dataset, whereas 'sag' and 'saga' are a better choice for larger data analysis as it takes less time for computation (du Boisberranger et al., n.d.).

The penalty is a regularization hyperparameter used in penalization, and it has a close connection with the solver. 'Newton-cg', 'sag' and 'lbfgs' solvers support only 'l2' penalties, while only 'saga' solver supports 'elasticnet' penalty. 'C' is the inverse of regularization strength which must be positive. A smaller value for 'C' indicates stronger regularization (du Boisberranger et al., n.d.).

Graph 5

Linear classification models



Source: compiled by the authors

Stochastic Gradient Descent (SGD)

SGD classifier is one of the effective linear classification methods applied to large datasets. The algorithm uses a first-order SGD learning routine. The parameter of the method is updated iteratively over training examples according to the formula:

$$\omega = \omega - \eta[\alpha\frac{\partial R(\omega)}{\partial \omega} + \frac{\partial L(\omega^T x_i + b, y_i)}{\partial \omega}]$$ (du Boisberranger et al., n.d.)

Where α is hyperparameter which controls regularization strength, *R* is regularization term that penalizes model complexity. *L* is loss function that measures model fit, ŋ is the learning rate and b is intercept which is updated similarly without regularization.

Gradient descent is one of the important algorithms used to minimize a cost function (Fuchs, 2019). In general, there are three popular types of gradient descent:

1. Batch gradient descent;
2. Stochastic gradient descent;
3. Mini-batch gradient descent.

Batch Gradient Descent computes the partial derivative of the cost function of an algorithm with regard to its parameters. In other words, the algorithm discovers how the different values of the model parameters impacts on cost function of the algorithm. The disadvantage is that it uses whole training data to compute it at every step. Therefore, the algorithm cannot handle large datasets (Geron, 2019).

Stochastic gradient descent addresses this problem as it picks random instances of training data and computes gradient descent based on that single instance. On the other hand, batch gradient descent will decrease the cost function smoothly, while it bounces up and down until the algorithm stops. The algorithm will not compute optimal parameter values (Geron, 2019).

Mini-batch gradient descent takes a small sample out of training data and computes batch gradient descent algorithm. Therefore, the cost function computation result is less erratic compared to stochastic gradient descent (Geron, 2019).

The method has many hyperparameters which makes it very complex. Important parameters such as loss function and regularization parameters such as penalty and alpha are considered in the tuning process of the model. The method with 'Hinge' loss function performs like linear SVM, while with 'log' function it gives logistic regression (du Boisberranger et al., n.d.).

The algorithm is very sensitive to feature scaling. But it is easy to implement and very efficient, especially for larger datasets (du Boisberranger et al., n.d.). Also, it continues to work without keeping the record in RAM (Fuchs, 2019).

Naïve Bayes classifier

In general, Naïve Bayes (NB) classifier is based on Bayes theorem assuming the independence of every feature:

$$P(y|x_1, \dots, x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1, \dots, x_n)}$$ (du Boisberranger et al., n.d.)

There are several types of NB classifiers. In this research, Gaussian Naïve Bayes (GaussianNB) classifier is used for the estimation of the likelihood of the features based on the formula:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} e^{-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}}$$ (du Boisberranger et al., n.d.)

where mean and standard deviation are estimated via maximum likelihood.

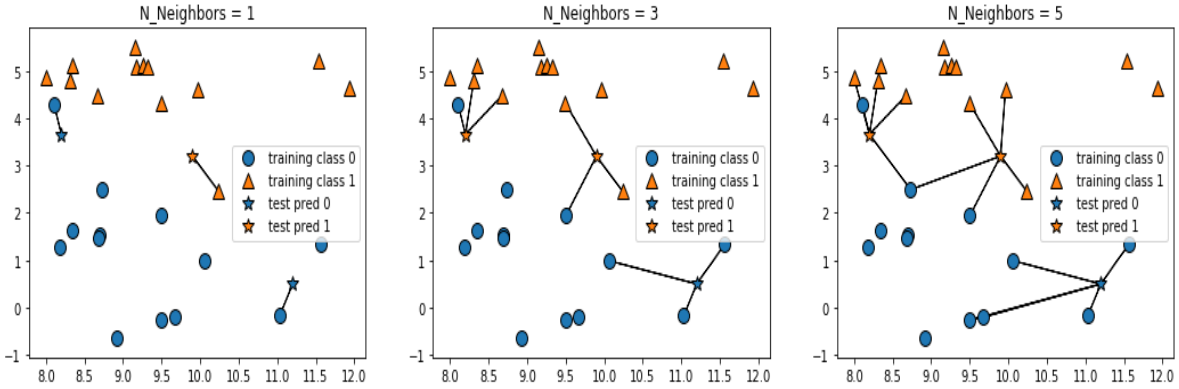
kth Nearest Neighbors

kNN is one of the simplest algorithms of supervised machine learning. The number of nearest neighbors (k) is the main parameter in this algorithm. The result of the new data will be identified based on the majority of the result of the nearest neighbors. The large value of k will overwhelm the effects of noise, but it will have on the boundary of the classification (du Boisberranger et al., n.d.). The distance is measured according to the Euclidean Distance formula:

$$d(x_i, x_j) = \|x_i - x_j\| = \left[(x_i - x_j)^T (x_i - x_j)\right]^{1/2}$$ (Brown & Mues, 2012)

Graph 6

kth Nearest Neighbors classification model



Source: Mglearn library (Mueller & Guido, 2017)

Decision Tree classifier

Decision Tree classifier simple algorithm that predicts target output via applying decision rules based on features data. The strength of the method is the ability to handle numerical, categorical features, and multi-output problems. The simplicity of the model is another advantage. But the model may create over-complex trees which lead to overfitting of the model (du Boisberranger et al., n.d.).

Criterion and a maximum depth of the tree are considered in the tuning process of the model. The first parameter is the function that measures the quality of a split. 'Gini' (Gini impurity) and 'entropy' (information gain) are two choices for criteria function (du Boisberranger et al., n.d.).

Random Forest Classifier

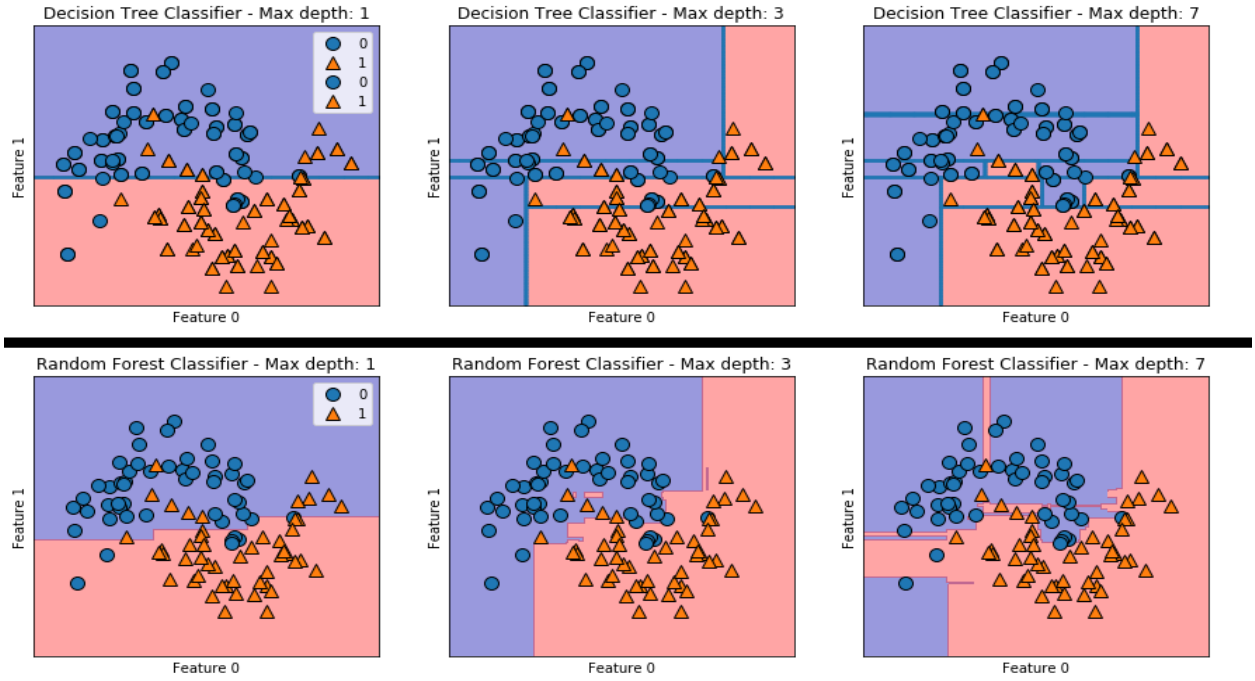
Random Forest Classifier is an ensemble classification method that has all the hyperparameters of the Decision Tree classifier. The difference between the two models is that the former searches for the best feature among a subset of features, while the latter seeks for the best feature when splits a node. Thus, a random forest classifier results in a bigger tree (Geron, 2019).

Decision tree classifier

Decision tree classification shows higher variance which leads to overfitting. Random forest classifier combines diverse trees, thereby achieves the reduction of the variance. Subsequently, it creates a better model than the decision tree classifier (du Boisberranger et al., n.d.).

Graph 7

Decision tree and random forest classifier on scikit-learn dataset



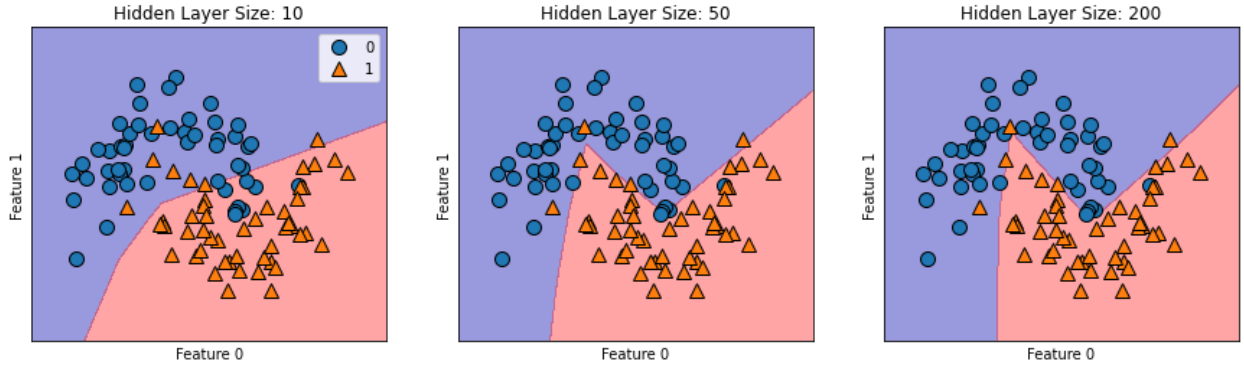
Source: compiled by the authors

Neural Network (Multilayer Perceptron)

A neural network is a model that was inspired by the human brain structure. The main structure of the algorithm consists of input, output and hidden layers. The size of hidden layers is an important hyperparameter of the algorithm. Each layer has nodes that hold a number, and each node receives a signal from each node from the previous layer and sends a signal to the nodes in the next layer. Each signal has weight and it goes along with a bias on which the input has no impact (Hansen, 2019).

Graph 8

Neural network classifier



Source: compiled by the authors

One important feature of the neural network is the gradient descent algorithm. It is used to minimize the deviation between the output and the calculated value of the output. The neural network uses backpropagation to calculate the gradients. Then, it updates all biases and adjusts weights of all nodes starting from the output (Hansen, 2019).

XGBoost

Extreme gradient boosting (XGBoost) is one of the widely used algorithms. It is an implementation of gradient boosted decision trees. The execution speed and model performance are the advantages over other gradient boosting algorithms (Brownlee, 2016).

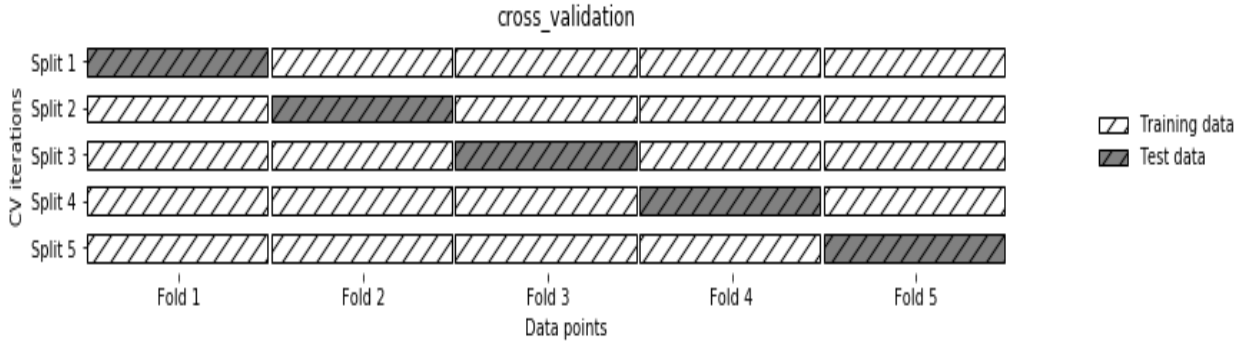
The advantage of the method is scalability. It uses a novel tree learning algorithm for handling sparse data and can handle missing values automatically. Parallel and distributed processing of the data makes it one the fastest algorithms. The algorithm uses out-of-core computation and processes millions of examples on a computer. Therefore, it is one of the best candidates to process large data. Another important feature of the method is called tree pruning. It results in deeper, but optimized trees (Chen & Guestrin, 2016).

Cross-validation

Cross-validation is a statistical method used to measure the performance of the methods based on training and testing dataset. A commonly used version of cross-validation is k-fold cross-validation, where the number of folds is 5 or 10 (Mueller & Guido, 2017). In the research, 5-fold cross-validation was used as it is shown in Graph 9. Consequently, the dataset is divided into five equal folds. If the first fold is used as a test set, the remaining folds as training set. To sum up, the goal is to compute the mean and standard deviation of the accuracies and analyze the impact of datasets on the model accuracy based on metrics.

Graph 9

Five-fold cross-validation



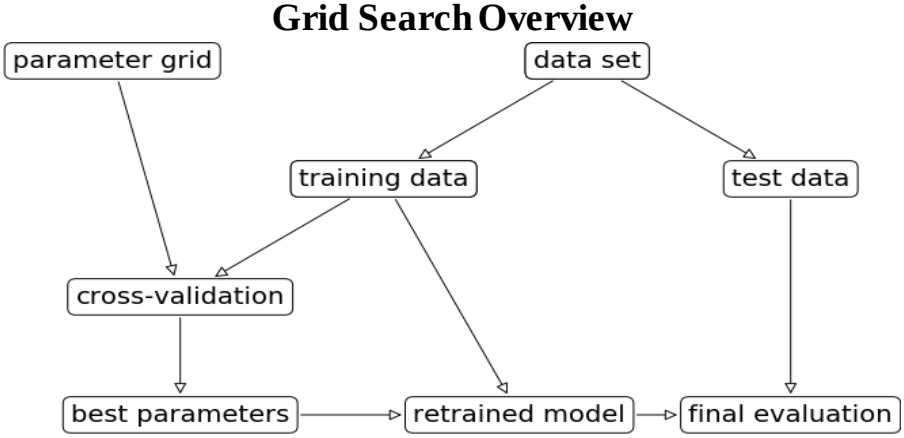
Source: Mglearn library (Mueller & Guido, 2017)

Grid Search

Grid Search is the final step of the model construction. In this step, the selected machine learning model is tuned. The parameter grid is the first step of the tuning process where a set of all possible hyperparameters of the algorithms is listed. Then,

the model with a different set of hyperparameters is applied to the training and test data. This step helps to identify the set of parameters by which the model achieves the highest score. The model will be reconstructed with the set of best hyperparameters (Mueller & Guido, 2017).

Graph 10



Source: Mglearn library (Mueller & Guido, 2017)

The big question in this step was to choose the hyperparameters to tune the model. The several models have many hyperparameters and it takes too much time to tune the process. Thereby, the parameter grid was constructed based on the most important parameters. For instance, the only the number of nearest neighbors was the only parameter selected for tuning the kNN model. Kaggle, the website for data scientists, provides a grid search model for some models that were used in the research.

Metrics

In machine learning, regression, clustering and classification algorithms use different performance metrics. Classification algorithms also have different metrics for binary and multiclass classification. There are 6 metrics used in this research:

1. Accuracy score
2. Precision score
3. Recall score
4. F1 score
5. Jaccard score
6. The area under the receiving operating characteristic (ROC) curve
7. Type 2 error percentage

Confusion matrix

The confusion matrix is a matrix table used to evaluate the performance of the classifier. It generally indicates the number of correct and incorrect results of a classification algorithm. It also gives information about the type 1 and type 2 errors which will be explained in this section.

Table 2

Confusion matrix

	Actual class (observation)	
Predicted class (expectation)	TP (true positive) Correct result	FP (false positive) Unexpected result
	FN (false negative) Missing result	TN (true negative) Correct absence of result

Source: Binary classification (du Boisberranger et al., n.d.)

1. Accuracy score

Accuracy is the most important metric in the research. It is also used as a scoring metric to tune the model in a grid search. The metric indicates the fraction of the correct results of the model:

$$Accuracy(y_{true}, y_{pred}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} 1(if\ y_{true} = y_{pred})(du\ Boisberranger\ et\ al., n.d.)$$

$$= \frac{TP + TN}{TP + FP + FN + TN}\ (Geron, 2019;\ Mueller\ \&\ Guido, 2017)$$

2. Precision score

The precision metric is the number of correct positive predictions divided by the number of total positive predictions.

$$Precision = \frac{TP}{TP + FP}\ (Geron, 2019; Mueller\ \&\ Guido, 2017)$$

3. Recall score

Recall is the number of correct positive predictions divided by the number of actual positive outcomes.

$$Recall = \frac{TP}{TP + FN}\ (Geron, 2019; Mueller\ \&\ Guido, 2017)$$

4. F1 score

Precision and recall are important metrics. However, none of them separately will not give full picture. F1 is another metric used for binary classification, and it is harmonic mean of precision and recall (Geron, 2019; Mueller & Guido; 2017).

$$F_1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = 2 \times \frac{precision \times recall}{precision + recall} = \frac{TP}{TP + \frac{FN + FP}{2}}\ (Geron, 2019)$$

5. Jaccard similarity coefficient score

Jaccard similarity score (JSC) is the intersection of actual and predicted output to their union.

$$JSC = \frac{TP}{TP + FP + FN} = \frac{F_1}{2 - F_1} \ (Labatut \ \& \ Cherifi, 2011)$$

6. Area under ROC curve

Receiving operating characteristic (ROC) curve is an important tool to analyze the performance of the binary classifiers. It is a line curve that plots the TP rate against the FP rate. The area under this curve is another way to compare the classifiers (Geron, 2019).

The value of these metrics ranges between 0 and 1: the higher the value – the more suitable the model for consumer credit analysis. In addition, these metrics have a very close relationship.

7. Type 2 percentage

Machine learning classifiers as statistical models also have type 1 and type 2 errors. Type 1 error is also known as false positive (FP) error (Schmarzo, 2018). For example, one of the second-tier commercial banks implements the models for credit scoring. The model rejects issuing a loan to a consumer as it declares that it will become a non-performing loan in the future. But, in reality, the consumer is capable of fully repaying loans. In this situation, banks reject to issue debt, but it will not have a significant impact on the liquidity and solvency of the bank.

False-negative (FN) is another type of error which is also known as type 2 error (Schmarzo, 2018). Suppose that the bank issues debt to the consumer based on the output of machine learning which claims that the consumer will not fail to repay it. In reality, the loan becomes non-performing and it has negative consequences to the liquidity of the bank. The higher percentage of type 2 error indicates how worse the model is.

4. Discussion of the results

As discussed before, the random undersampling and oversampling techniques are not the best resampling technics. The former leads to loss of information, while the latter leads to overfitting problems. In addition, random undersampling was the only adaptable option for regulatory data in this research. In this section, the result of all classifiers will be discussed based on the data on which they were applied.

Undersampling data

According to the outcomes of the exercises, linear classifiers and the Naïve Bayes classifier are not the best options for credit scoring. This shows that linear models and Naïve Bayes cannot be a solution for credit scoring problems. Table 3 indicates that all classifiers had underfitting problems: the algorithms did not model training data well and performed inadequately on testing data.

Table 3

Accuracy of classifiers applied to undersampled data

	Linear Models		Non-Linear Models					
	Logistic Regression	SGD	Naïve Bayes	kNN	Decision Tree	Random Forest	Neural Networks	XGB
Training	58,6%	58,7%	59,0%	68,6%	68,3%	64,9%	70,8%	69,6%
Testing	58,7%	58,9%	59,0%	67,1%	65,9%	64,6%	70,0%	67,2%

Source: compiled by the authors

In spite of weak performance, the stochastic gradient descent (SGD) classifier gave one of the lowest percentages of type 2 error. Neural Networks classifier generated the best result among the models applied to undersampled data based on most of the metrics. But a high percentage of type 2 error does not make it a favorite. Also, the model takes comparatively more time for computation. Thereby, the extreme gradient boosting (XGB) classifier is the best option, because it is fast, accurate, precise, and also had the second highest accuracy score and second-lowest percentage of type 2 error.

Table 4

Models and performance results

		Metrics						
		Accuracy	*Precision*	*Recall*	*F1*	*JSC*	*AUC_ROC*	*Type* 2 error
Linear	Logistic Regression	58,7%	59,0%	58,5%	58,7%	41,6%	58,7%	20,8%
	SGD	58,9%	57,9%	66,5%	61,9%	44,8%	58,9%	16,8%
Non-Linear	Naïve Bayes	59,0%	60,1%	54,3%	57,0%	39,9%	59,0%	23,0%
	kNN	67,1%	68,4%	64,0%	66,1%	49,4%	67,1%	18,6%
	Decision Tree	65,9%	66,9%	63,5%	65,2%	48,3%	65,9%	18,3%
	Random Forest	64,6%	63,5%	69,4%	66,3%	49,6%	64,6%	15,4%
	Neural Networks	70,0%	72,8%	64,1%	68,2%	51,7%	70,0%	18,0%
	XGB	67,2%	67,4%	67,3%	67,4%	50,8%	67,2%	16,4%

Source: compiled by the authors

Oversampled data

The classifiers performed much better on oversampled data. One of the reasons for the success is the SMOTE resampling technique which brought additional information to the data. Linear classifiers and Naïve Bayes classifiers

showed the worst performance results. Other classifiers well-fitted training data and performed well on oversampled data.

Table 5

Accuracy of classifiers applied to oversampled data

	Linear Models		Non-Linear Models					
	Logistic Regression	SGD	Naïve Bayes	kNN	Decision Tree	Random Forest	Neural Networks	XGB
Training	64,1%	64,1%	64,9%	99,9%	76,8%	99,6%	73,6%	74,6%
Testing	64,1%	64,1%	64,9%	83,5%	75,3%	84,8%	73,6%	74,4%

Source: compiled by the author

Neural Networks was a promising model, but there is an insignificant difference between the model performance on undersampled and oversampled data. Despite of an increase in the accuracy on oversampled data in comparison with undersampled data, the Neural Networks could not outperform non-linear models.

According to Table 6, kNN and random forest classifiers outperformed other classifiers based on all indicators. In addition, both models demonstrated the lowest percentage of type 2 error. However, the random forest classifier outperformed the kNN classifier by all metrics except precision.

Table 6

Models and performance results

		Metrics						
		Accuracy	*Precision*	*Recall*	*F1*	*JSC*	*AUC_ROC*	*Type* 2 error
Linear	Logistic Regression	64,1%	63,9%	65,1%	64,5%	47,6%	64,1%	17,5%
	SGD	64,1%	63,9%	65,1%	64,5%	47,6%	64,1%	17,5%
Non-Linear	Naïve Bayes	64,9%	62,6%	74,2%	67,9%	51,4%	64,9%	12,9%
	kNN	83,5%	85,2%	81,0%	83,0%	71,0%	83,5%	9,5%
	Decision Tree	75,3%	74,8%	76,2%	75,5%	60,7%	75,3%	11,9%
	Random Forest	84,8%	84,7%	85,0%	84,8%	73,6%	84,8%	7,5%
	Neural Networks	73,6%	71,2%	79,1%	75,0%	60,0%	73,6%	10,5%
	XGB	74,4%	73,5%	76,2%	74,9%	59,8%	74,4%	11,9%

Source: compiled by the authors

5. Conclusion

The results of the study showed that the machine learning models work sufficiently well on the basis of regulatory data collected by the central bank. In addition, the analysis of consumer loans with machine learning algorithms demonstrated a unique insight regarding the features of (bad and good) consumer borrowers as well as provided information to check the accuracy of issued consumer loans by second-tired banks.

In this research, we have presented that it is critical to perform data quality checks (during arrangement and cleaning procedures to eliminate unnecessary variables), and it is essential to deal with an imbalanced set of training data to avert bias in favor of most categories.

In terms of forecasting accuracy of models, oversampled data adjusted by the SMOTE method showed more promising results in comparison with randomly undersampled data. In other words, the oversampled data adjusted by SMOTE helped to minimize the loss of information and increase the forecasting power. Models with well-fitted training data performed better on oversampled data compared with randomly undersampled data.

Furthermore, the non-linear models illustrated more accurate predictions compared with the linear models. Specifically, the non-linear models such as the random forest and kNN classifiers on oversampled data outperformed other classification models, on the other hand, the linear models such as logistic regression and SGD classifier showed the weakest results among compared eight models.

To sum up, the models based on regulatory data can be an adequate foundation for the evaluation of credit risk of issued consumer loans by second-tier banks, and also can help the central bank to predict potential systematic risks. So, according to outcomes of the study, an assessment of credit risk through machine learning can be a good supplement for regulation of the second-tier banks that issue consumer loans.

In order to improve the quality of this research several steps should be done:

Additional data: According to Grier (2012), there are some additional data such as incomes, social status, experience, education, and the sector in which a borrower works, which might bring a positive impact on the performance of algorithms.

In terms of models, there are some approaches that are able to improve their accuracy:

a. Application of hybrid machine learning methods

b. Building a Selective Combined Forecasting System

c. Including additional parameters in grid search and over/undersampling

References

Addo, P., Guegan, D., & Hassani, B. (2018). Credit Risk Analysis Using Machine and Deep Learning Models. *Risks*, *6*(2), 38. doi: 10.3390/risks6020038

Alencar, R. (2017). Resampling strategies for imbalanced datasets. Retrieved from https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets

Baesens, B., Van Gestel, T., Viaene, S., Stepanova, M., Suykens, J., & Vanthienen, J. (2003). Benchmarking state-of-the-art classification algorithms for credit scoring. *Journal of The Operational Research Society*, *54*(6), 627-635. doi: 10.1057/palgrave.jors.2601545

Brown, I., & Mues, C. (2012). An experimental comparison of classification algorithms for imbalanced credit scoring data sets. *Expert Systems with Applications*, *39*(3), 3446-3453. doi: 10.1016/j.eswa.2011.09.033

Brownlee, J. (2016). A Gentle Introduction to XGBoost for Applied Machine Learning. Retrieved from https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/

Brownlee, J. (2016). Metrics to Evaluate Machine Learning Algorithms in Python. Retrieved from https://machinelearningmastery.com/metrics-evaluate-machine-learning-algorithms-python/

Brownlee, J. (2020). SMOTE for Imbalanced Classification with Python. Retrieved from https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

Chawla, N., Bowyer, K., Hall, L., & Kegelmeyer, W. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16, 321-357. Doi: 10.1613/jair.953

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of The 22Nd ACM SIGKDD International Conference On Knowledge Discovery and Data Mining*, 785–794. doi: 10.1145/2939672.2939785

du Boisberranger, J., Van den Bossche, J., Esteve, L., Fan, T., Gramfort, A., & Grisel, O. et al. User guide: contents — scikit-learn 0.23.2 documentation. Retrieved from https://scikit-learn.org/stable/user_guide.html

Fuchs, M. (2019). Introduction to SGD Classifier - Michael Fuchs Python. Retrieved from https://michael-fuchs-python.netlify.app/2019/11/11/introduction-to-sgd-classifier/

Geron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (2nd ed.). Sebastopol, California: O'Reilly Media, Incorporated.

Grier, W. (2012). *Credit analysis of financial institutions* (2nd ed., pp. 294-296). London: Euromoney Institutional Investor PLC.

Hansen, C. (2019). Neural Networks: Feedforward and Backpropagation Explained. Retrieved from https://mlfromscratch.com/neural-networks-explained/#overview

Henley, W., & Hand, D. (1996). A k-Nearest-Neighbour Classifier for Assessing Consumer Credit Risk. *The Statistician*, *45*(1), 77. doi: 10.2307/2348414

Labatut, V., & Cherifi, H. (2011). Evaluation of Performance Measures for Classifiers Comparison. *Ubiquitous Computing and Communication Journal*, *6*, 21-34. Retrieved from https://arxiv.org/abs/1112.4133

Lemaitre, G., Nogueira, F., Oliveira, D., & Aridas, C. 2. Over-sampling — imbalanced-learn 0.5.0 documentation. Retrieved from https://imbalanced-learn.readthedocs.io/en/stable/over_sampling.html#smote-adasyn

Lessmann, S., Baesens, B., Seow, H., & Thomas, L. (2015). Benchmarking state-of-the-art classification algorithms for credit scoring: An update of research. *European Journal of Operational Research*, *247*(1), 124-136. doi: 10.1016/j.ejor.2015.05.030

Mueller, A., & Guido, S. (2017). *Introduction to Machine Learning with Python* (1st ed.). Sebastopol, California: O'Reilly Media.

Munkhdalai, L., Munkhdalai, T., Namsrai, O., Lee, J., & Ryu, K. (2019). An Empirical Comparison of Machine-Learning Methods on Bank Client Credit Assessments. *Sustainability*, *11*(3), 699. doi: 10.3390/su11030699

Schmarzo, B. (2018). Understanding Type 1 and Type 2 Errors [Blog]. Retrieved from https://www.datasciencecentral.com/profiles/blogs/understanding-type-i-and-type-ii-errors

Tsai, C., & Chen, M. (2010). Credit rating by hybrid machine learning techniques. *Applied Soft Computing*, *10*(2), 374-380. doi: 10.1016/j.asoc.2009.08.003

Appendix

```
# Important libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
import os
path = 'E:\...'  #selecting path to document where the file is located
os.chdir(path)
df_ml = pd.read_csv('filename.csv')
X_df = df_ml.iloc[:,:-1].values
y_df = df_ml.iloc[:,-1].values

# Random Undersampling algorithm from imblearn library
from imblearn.under_sampling import RandomUnderSampler as rus
us = rus(random_state=42)
X, y = us.fit_resample(X_df, y_df)

# SMOTE-NC oversampling algorithm from imblearn library
from imblearn.over_sampling import SMOTENC
sm = SMOTENC(random_state=42, categorical_features=[0,1,2,3,4,5])  # there column of the data are categorical variables
X, y = sm.fit(X_df, y_df)

# Encoding categorical data
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder

# the first column of categorical variables contained 18 different variables (three cities and regions (also former name for one region)), thus create 18 rows
ct0 = ColumnTransformer(transformers=[('encoder', OneHotEncoder(),[0])], remainder='passthrough')
X=np.array(ct0.fit_transform(X))

# the second column of categorical variables contained 4 different variables (four type of currency, in which a loan was issued), thus create 4 rows
ct1 = ColumnTransformer(transformers=[('encoder', OneHotEncoder(),[18])], remainder='passthrough')
X=np.array(ct1.fit_transform(X))

# the third column of categorical variables contained 2 different variables (credit card or cash), thus create 2 rows
ct2 = ColumnTransformer(transformers=[('encoder', OneHotEncoder(),[18])], remainder='passthrough')
```

```
X=np.array(ct2.fit_transform(X))

# the fourth column of categorical variables contained 2 different variables (for consumer use or autoloan), thus create 2 rows
ct3 = ColumnTransformer(transformers=[('encoder', OneHotEncoder(),[18])], remainder='passthrough')
X=np.array(ct3.fit_transform(X))

# the fifth column of categorical variables contained 2 different variables (male or female), thus create 2 rows
ct4 = ColumnTransformer(transformers=[('encoder', OneHotEncoder(),[18])], remainder='passthrough')
X=np.array(ct4.fit_transform(X))

# the sixth column of categorical variables contained 2 different variables (local or foreigner), thus create 2 rows
ct5 = ColumnTransformer(transformers=[('encoder', OneHotEncoder(),[18])], remainder='passthrough')
X=np.array(ct5.fit_transform(X))

# Label Encoder of y variable
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)

# splitting data into train and test dataset
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, random_state=1)

# feature scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train[:,-3:] = sc.fit_transform(X_train[:,-3:])
X_test[:,-3:] = sc.transform(X_test[:,-3:])

# Principal component analysis (PCA)
from sklearn.decomposition import PCA
pca = PCA(n_components=a)  # a is quantity where cumulative explained variance ratio > 95%
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

# Logistic Regression
```

```
from sklearn.linear_model  import LogisticRegression
log = LogisticRegression().fit(X_train,y_train)
y_tr_log_pred = log.predict(X_train)
y_ts_log_pred = log.predict(X_test)

# Stochastic Gradient Descent Classifier
from sklearn.linear_model  import SGDClassifier
sgd=SGDClassifier().fit(X_train,y_train)
y_tr_sgd_pred = sgd.predict(X_train)
y_ts_sgd_pred = sgd.predict(X_test)

# Gaussian Naïve Bayes Classifier
from sklearn.naive_bayes  import GaussianNB
nbc = GaussianNB().fit(X_train,y_train)
y_tr_nb_pred = nbc.predict(X_train)
y_ts_nb_pred = nbc.predict(X_test)

# kth nearest neighbors (kNN) Classifier
from sklearn.neighbors  import KNeighborsClassifier
knn=KNeighborsClassifier().fit(X_train,y_train)
y_tr_knn_pred = knn.predict(X_train)
y_ts_knn_pred = knn.predict(X_test)

# Decision Tree Classifier
from sklearn.tree  import DecisionTreeClassifier
dtc=DecisionTreeClassifier(max_depth=14,
criterion='entropy').fit(X_train,y_train)
y_tr_dt_pred = dtc.predict(X_train)
y_ts_dt_pred = dtc.predict(X_test)

# Random Forest Classifier
from sklearn.ensemble  import RandomForestClassifier
rfc=RandomForestClassifier().fit(X_train,y_train)
y_tr_rf_pred = rfc.predict(X_train)
y_ts_rf_pred = rfc.predict(X_test)

# Multi-layer  perceptron Classifier  (Neural  network)
from sklearn.neural_network  import MLPClassifier
nnc = MLPClassifier().fit(X_train,y_train)
y_tr_nnc_pred = nnc.predict(X_train)
y_ts_nnc_pred = nnc.predict(X_test)

# Extreme  gradient  boosting (XGB) Classifier
from xgboost import XGBClassifier
```

```
xgb = XGBClassifier().fit(X_train,y_train)
y_tr_xgb_pred = xgb.predict(X_train)
y_ts_xgb_pred = xgb.predict(X_test)

# Cross-validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = xgb, X = X_train, y = y_train, cv = 5)
print('Accuracy: {:.2f} %'.format(accuracies.mean()*100))
print('Standard deviation: {:.2f} %'.format(accuracies.std()*100))

### Note: we can put other models instead of XGB to analyze cross-validation result

# Grid Search for Logistic Regression
from sklearn.model_selection import GridSearchCV
parameters = [{'penalty': ['none'], 'solver':['newton-cg', 'sag', 'saga', 'lbfgs']},
              {'penalty': ['elasticnet'], 'C': [0.01, 0.1, 0.25, 0.5, 0.75, 1, 5, 10],
'solver':['saga']},
              {'penalty': ['l2'], 'C': [0.01, 0.1, 0.25, 0.5, 0.75, 1, 5, 10], 'solver':['newton-
cg', 'sag', 'saga', 'lbfgs']}]
grid_search = GridSearchCV(estimator = log,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 5,
                           n_jobs = -1)
grid_search.fit(X_train, y_train)
best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_
print('Best accuracy: {:.2f} %'.format(best_accuracy*100))
print('Best parameters: ',best_parameters)

# Grid Search for SGD Classifier
from sklearn.model_selection import GridSearchCV
parameters = [{"loss" : ["hinge", "log", "squared_hinge", "modified_huber"],
               "alpha" : [0.0001, 0.001, 0.01, 0.1], "penalty" : ["l2", "l1", "none"]}]
grid_search = GridSearchCV(estimator = sgd,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 5,
                           n_jobs = -1)
grid_search.fit(X_train, y_train)
best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_
print('Best accuracy: {:.2f} %'.format(best_accuracy*100))
print('Best parameters: ',best_parameters)
```

```
# Grid Search for Gaussian Naïve Bayes Classifier
from sklearn.model_selection import GridSearchCV
parameters = [{'var_smoothing': [1e-12,1e-10,1e-7,1e-4,1e-3,1e-2,1e-1,1,10]}]
grid_search = GridSearchCV(estimator = nbc,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 5,
                           n_jobs = -1)
grid_search.fit(X_train, y_train)
best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_
print('Best accuracy: {:.2f} %'.format(best_accuracy*100))
print('Best parameters: ',best_parameters)

# Grid Search for kNN Classifier
from sklearn.model_selection import GridSearchCV
parameters = [{'n_neighbors': list(range(1, 81))}]
grid_search = GridSearchCV(estimator = knn,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 5,
                           n_jobs = -1)
grid_search.fit(X_train, y_train)
best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_
print('Best accuracy: {:.2f} %'.format(best_accuracy*100))
print('Best parameters: ',best_parameters)

# Grid Search for Decision Tree Classifier
from sklearn.model_selection import GridSearchCV
parameters = [{'criterion':['gini', 'entropy'],'max_depth': list(range(1,21))}]
grid_search = GridSearchCV(estimator = dtc,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 5,
                           n_jobs = -1)
grid_search.fit(X_train, y_train)
best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_
print('Best accuracy: {:.2f} %'.format(best_accuracy*100))
print('Best parameters: ',best_parameters)

# Grid Search for Random Forest Classifier
```

```
from sklearn.model_selection import GridSearchCV
parameters = [{'n_estimators': list(range(1,21)), 'max_features': ['auto', 'sqrt', 'log2'],
            'max_depth' : ['None',8], 'criterion' :['gini', 'entropy']}]
grid_search = GridSearchCV(estimator = rfc,
                        param_grid = parameters,
                        scoring = 'accuracy',
                        cv = 5,
                        n_jobs = -1)
grid_search.fit(X_train, y_train)
best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_
print('Best accuracy: {:.2f} %'.format(best_accuracy*100))
print('Best parameters: ',best_parameters)

# Grid Search for MLP Classifier
from sklearn.model_selection import GridSearchCV
parameters = [{'hidden_layer_sizes':[100,200,300,[200,50],[100,100],[200,100]],
            'activation':['identity','logistic','tanh','relu'],
            'solver': ['adam'],
            'learning_rate':['constant','invscaling','adaptive'],
            'max_iter': [1000,1500,2000 ]}]
grid_search = GridSearchCV(estimator = nnc,
                        param_grid = parameters,
                        scoring = 'accuracy',
                        cv = 5,
                        n_jobs = -1)
grid_search.fit(X_train, y_train)
best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_
print('Best accuracy: {:.2f} %'.format(best_accuracy*100))
print('Best parameters: ',best_parameters)

# Grid Search for XGB Classifier
from sklearn.model_selection import GridSearchCV
parameters = [{'n_estimators': [1000], #number of trees, change it to 1000 for better
results
            'max_depth': [6,7,8],
            'learning_rate': [0.05], #so called `eta` value
            'objective':['binary:logistic'],
            'tree_method':['exact'],
            'min_child_weight': [11],
            'subsample': [0.8],
            'colsample_bytree': [0.7]}]
grid_search = GridSearchCV(estimator = xgb,
```

```
                    param_grid = parameters,
                    scoring = 'accuracy',
                    cv = 5,
                    n_jobs = -1)
grid_search.fit(X_train, y_train)
best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_
print('Best accuracy: {:.2f} %'.format(best_accuracy*100))
print('Best parameters: ',best_parameters)
```

After grid search, all best parameters should be implemented to be sure that the model with the best accuracy give higher accuracy and after that other metrics should be analyzed

```
# Example where all metrics that were used
from sklearn.metrics import confusion_matrix, accuracy_score
knn_cm_tr = confusion_matrix(y_train,y_tr_knn_pred)
print(knn_cm_tr)
accuracy_score(y_train, y_tr_knn_pred)

knn_cm_ts = confusion_matrix(y_test,y_ts_knn_pred)
print(knn_cm_ts)
accuracy_score(y_test, y_ts_knn_pred)

from sklearn.metrics import roc_auc_score, jaccard_score, f1_score,
precision_score, recall_score
print(roc_auc_score(y_test,y_ts_knn_pred))
print(jaccard_score(y_test,y_ts_knn_pred))
print(f1_score(y_test,y_ts_knn_pred))
print(precision_score(y_test,y_ts_knn_pred))
print(recall_score(y_test,y_ts_knn_pred))
```

Here, the metrics were used to analyze the performance of kNN Classifier, the variables should be changed to achieve the result of the other models