MEF UNIVERSITY

DEFAULT PREDICTION MODELS FOR MORTGAGE LOANS

Capstone Project

Ilknur Tezgiden

İSTANBUL, 2018

MEF UNIVERSITY

DEFAULT PREDICTION MODELS FOR MORTGAGE LOANS

Capstone Project

Ilknur Tezgiden

Advisor: Dr. Berk Orbay

İSTANBUL, 2018

MEF UNIVERSITY

Name of the project: Default Prediction Models For Mortgage Loans
Name/Last Name of the Student: Ilknur Tezgiden
Date of Thesis Defense: 03/09/2018

I hereby state that the graduation project prepared by Ilknur Tezgiden has been completed under my supervision. I accept this work as a “Graduation Project”.

03/09/2018
Dr. Berk Orbay

I hereby state that I have examined this graduation project by Ilknur Tezgiden which is accepted by his supervisor. This work is acceptable as a graduation project and the student is eligible to take the graduation project examination.

03/09/2018

Prof. Özgür Özlük

Director
of
Big Data Analytics Program

We hereby state that we have held the graduation examination of and agree that the student has satisfied all requirements.

THE EXAMINATION COMMITTEE

Committee Member	Signature
1. Dr. Berk Orbay	………………………..
2. Prof. Özgür Özlük	………………………..

Academic Honesty Pledge

I promise not to collaborate with anyone, not to seek or accept any outside help, and not to give any help to others.

I understand that all resources in print or on the web must be explicitly cited.

In keeping with MEF University's ideals, I pledge that this work is my own and that I have neither given nor received inappropriate assistance in preparing it.

Name	Date	Signature
Ilknur Tezgiden	03/09/2018	

EXECUTIVE SUMMARY

DEFAULT PREDICTION MODELS FOR MORTGAGE LOANS

Ilknur Tezgiden

Advisor: Dr. Berk Orbay

AUGUST 2018, 79 pages

The mortgage financial crisis which in U.S. mid 2000’s has been expanded and took hold of the other countries in a short time. The impact of the crisis forced financial institutions, especially the banks, to monitor their credit portfolio closely. The aim of this study is to develop models for predicting mortgage default cases in the loan life cycle. Those models were developed by using 50.000 loan repayments that was randomly selected between 60 months. Total 622489 observation and 23 features were there in the dataset. Classification algorithms were applied on the models since the expected outputs of the models were either default (1) or not-default (0).

Key Words: Mortgage Loans, Default Prediction, Supervised Learning, Regression Models

ÖZET

EV KREDİLERİ İCİN BATIK TAHMİN MODELLERİ

İlknur Tezgiden

Tez Danışmanı: Dr. Berk Orbay

AĞUSTOS, 2018, 79 sayfa

2000' li yılların ortalarında ABD'de yaşanan ve ev kredilerinden kaynaklanan finansal kriz kısa süre içinde dünya çapında etkilerini göstermiştir. Yaşanan bu krizin etkileri finansal kurumları, özellikle bankaları, kredi portföylerinin kalitesini daha yakından izleme zorunda bırakmıştır. Bu çalışmanın amacı, finansal kurumlar için, müşteriye tahsis edilen ipotek karşılıklı ev kredisinin vadesi içerisinde batıp batmayacağını tahminleyecek modeller oluşturmaktır. İlgili modeller için 50.000 kredinin rastgele seçilen geri ödeme aylarından oluşan veri seti kullanılmıştır. Veri setinde toplam 622489 adet gözlem ile ilgili kredilere ait 23 özellik bulunmaktadır. Kredi tahsis edilen müşterilerin batıp (1) batmayacağı (0) sorusuna sınıflandırma modelleri kullanılarak cevap aranmıştır.

Anahtar Kelimeler: Ev Kredileri, Batık Tahmini, Güdümlü Öğrenme, Sınıflandırma Modelleri

TABLE OF CONTENTS

1. INTRODUCTION

The recent mortgage crisis in mid 2000's was characterized by a fall in house prices, an increase in mortgage defaults and home foreclosures, and a decrease in the value of mortgage-backed securities. These events initially affected residential construction and the financial sector, but their negative effects spread quickly to other sectors of the economy. Foreclosures appear also to have had negative feedback effects on the values of neighboring properties, worsening the decline in house prices (Campbell, Giglio, and Pathak 2011). In the Figure-1, the depth of the crisis in U.S. can be seen clearly.

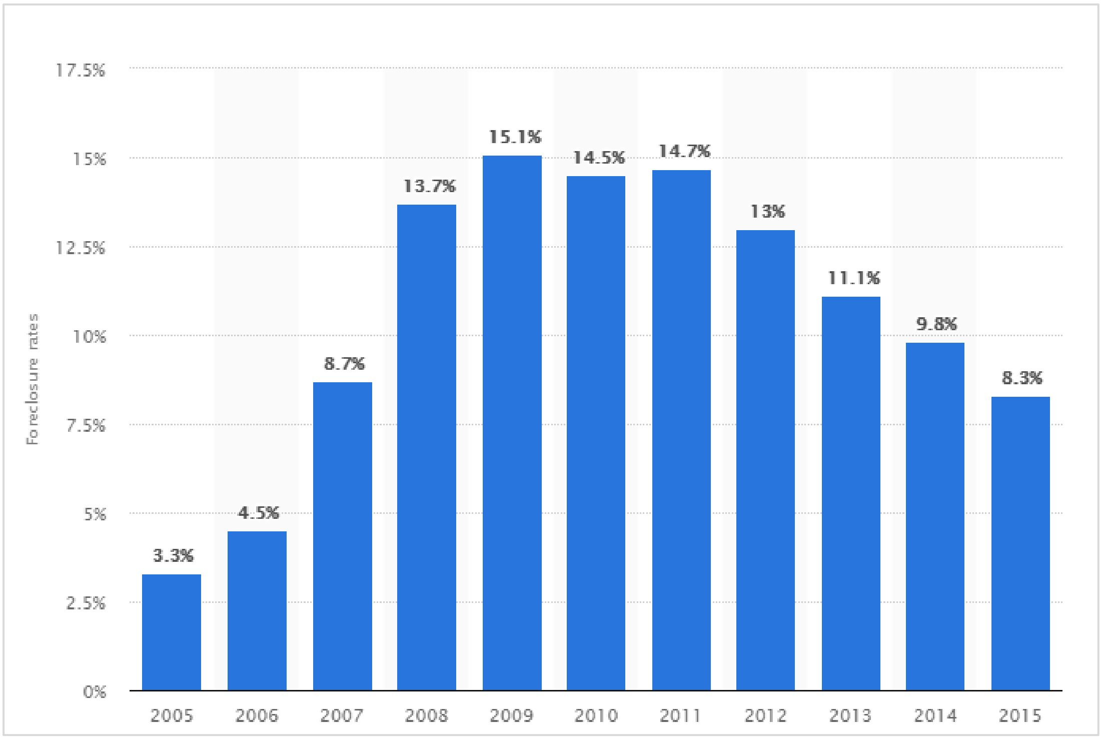


Figure-1: The foreclosure rate in the US between 2005-2015[1]

This crisis highlighted the importance of risk measurement and monitoring for financial institutions not only in the U.S. but also in all other countries. Moreover, financial institutions, which have deterioration on credit portfolio, faced the risk of inefficient use of their financial sources because of increasing capital and provision obligation. Because of the above reasons, financial institutions were more interested in the customers' ability to pay their debt and started to use default prediction models to detect their customers' risks before default cases.

[1]www.statista.com/statistics/206014/us-foreclosure-rates-on-subprime-conventional-loans-since-2000

The dataset I used in the project composed of the information about the loan and economic outlook when the loan allocated, and its installments paid. Because the developed models reflect the changes in customers' situation and economic outlook during the loan life cycle, the model's outputs can be used as a behavioral scorecard for the monitoring of the customers' default risks. In the dataset there are 23 features detailed below:

- Id: Borrower id
- Time: Time stamp of observation
- Origination time: Time stamp for origination
- First time: Time stamp for first observation
- Maturity time: Time stamp for maturity
- Balance time: Outstanding balance at observation time
- LTV time: Loan-to-value ratio at observation time, in %
- Interest rate time: Interest rate at observation time, in %
- HPI time: House price index at observation time, base year = 100
- GDP time: Gross domestic product (GDP) growth at observation time, in %
- Unemployment time: Unemployment rate at observation time, in %
- Retype originate time: Real estate type condominium = 1, otherwise = 0
- Retype PU originate time: Real estate type planned urban development = 1, otherwise= 0
- Retype SF originate time: Single-family home = 1, otherwise = 0
- Investor originate time: Investor borrower = 1, otherwise = 0
- Balance originate time: Outstanding balance at origination time
- FICO originate time: FICO score at origination time, in %
- LTV originate time: Loan-to-value ratio at origination time, in %
- Interest rate originate time: Interest rate at origination time, in %
- HPI originate time: House price index at origination time, base year = 100
- Default time: Default observation at observation time
- Payoff time: Payoff observation at observation time
- Status time: Default (1), payoff (2), and nondefault/ non-payoff (0) observation at observation time

2. LITERATURE REVIEW

2.1 Literature

In this context, machine learning algorithms have been started to use by financial institutions more efficiently in the last decades, but the studies related to the analysis of the default cases have been started from 1960's. It was seen that the first-generation studies focused on the default decisions with lender review. The studies were based on small sample data and variables (Quercia & Stegman, 1992). For the instances, the study about to the relationship between loan age and loan-to-value ratio on mortgage default rates (Furstenberg, 1970), default rate and delinquency rate analysis (Morton, 1975) and the potential delinquency based on the analysis of the income variability (Webb, 1982) can be counted.

Second-generation studies changed their point of view and focus on the borrower and the other variables like home prices, crime rate, interest rate, etc. and the relationship between them were analyzed (Vandell, 1996); (Rosenblatt & Crawford, 1999); (Deng, Quigley, & Van Order, 2003); (Bajari & Chu, & Park, 2008); (Gerardi & Willen, 2009). In 2010s, the relationship between sustainability features and default risk (transportation-, location-, and affordability) in multifamily housing was highlighted to by Pivo (2014).

In the late 1990's the machine learning algorithms were commenced on more often for mortgage default prediction. Machine learning algorithms' effect on financial institutions when deciding whether or not to grant credit to consumers was examined by Lyn, (2000). The usage of neural networks for credit scoring was investigated by West (2000). Khandani (2010) surveyed the machine learning algorithms' impact on diminishing the credit losses. In order to evaluate the performance of a number of machine learning algorithms for mortgage default status a study performed by Fitzpatrick, (2016) and (Akindaini, 2017).

In addition, due to the nature of the financial transactions, it is expected that the number of the default cases is much more lower than the number of the non- default cases. Because of this, the dataset obtained from loan transactions is always imbalanced. This issue was researched by Provost (2000) and Han & WY & Mao (2005).

2.2 Machine Learning Algorithms

2.2.1 Tree-Based Algorithms

Tree-based algorithms (such as decision tree) are used for both regression and classification problems. Regression trees should be used when dependent variable is continuous, otherwise (dependent variable is categorical) classification trees should be considered. Tree-based algorithms are easy to understand and interpret. These algorithms are fast to identify the significant variables and relations between them. Outliers affect tree-based algorithms less than other algorithms and these algorithms work well both numeric and categorical variables. Although there are mentioned advantages, tree-based algorithms are tending to be over-fitting and information lost may be seen when working with continuous variables.

Even if these algorithms have some advantages mentioned above, overfitting and high variance is the most important problems of them for the machine learning algorithms. In order to overcome these problems in real-world Random Forest, Boosting and Bagging Algorithms are preferred usually. Each of these algorithms tries to minimize the variance or overfitting and often provides a development of the model predictive power.

2.2.2 Boosting, Bagging and Random Forest Algorithms

Boosting is an example for ensemble approach and consists of two steps. In the first step the model takes all the distributions and assign equal weight to them. In case of misclassification in the first model, these incorrect observations are assigned to next model with higher attention or weight to classify correctly. This process continues until the model reaches the higher predictive power possible. Boosting algorithms (AdaBoost, Gradient Tree Boosting and XGBoost) are more interpretable than bagged trees/random forest and working with the categorical features are easy. On the other hand, these algorithms tend to be overfit when number of trees is increasing.

Bagging (stands for Bootstrap Aggregating) aims to decrease the variance of the model by generating additional data for training from the original dataset and works well high variance -low bias models. According to Leo Breiman who wrote a paper in 1996 called "Bagging Predictors" bagging as:

"Bagging predictors is a method for generating multiple versions of a predictor and using these to get an aggregated predictor."

Bagging algorithms provide better results for overfitting but interpretation of the model is hard, and in the existence of correlation between features can not decrease the variance in desired level.

In Random Forest Algorithms (also in Extremely Randomized Tree) many classification trees are grown. Each tree gives a classification, and this classification measures with "votes". Over all the trees in the forest the most trees are chosen as classification.

Random Forest algorithm is easy to use. Moreover, the number of hyperparameters is not high and they are simple to understand, and it prevents overfit. On the other hand, this algorithm can be slow with a large number of trees.

2.2.3 Naïve Bayes Algorithms

Naïve Bayes Algorithms are based on Bayes' Theorem and a simple probabilistic classifier. This algorithm is easy, fast and performs well in multiclass prediction. This algorithm is more suitable for categorical variables than numeric. On the other hand, in real-life examples it is hard to find independent predictors. Besides, if the training dataset does not cover all real-life examples for a categorical variable, unknown example can be assigned "0" and model cannot be able to prediction for this example.

2.2.4 K-Neighbors

K-Neighbors classification is a type of instance-based learning. This algorithm does not develop an internal model only it stores instance of the training data. To do this, the distance for each observation is voted to nearest neighbors and according to simple majority vote the class of nearest neighbor is assigned. K-Neighbors algorithm uses both supervised and unsupervised learning and provides high accuracy. This algorithm is insensitive to outliers, works well with numeric and categoric variables, and there is no need make assumptions about data. On the other hand, working this algorithm requires a lot memory because of this reason it is computationally expensive.

2.2.5 Logistic Regression

Logistic Regression Algorithms use maximum-likelihood in order to estimate the coefficients (beta values b) from training data. This algorithm does not require too many computational resources, input features to be scaled and require any tuning. It is also easy to interpret, to regularize and to implement. But when the problem is not linear, logistic regression cannot solve the problem properly. Besides, logistic regression does not work well when variables have normal distribution, categoric variables have too many category, inadequate examples for predicted parameters, and the classes are well separated. In order to overcome the weakness of the Logistic Regression, Linear Discriminant Analysis (LDA) and Quadratic Discriminant Analysis (QDA) are generally used.

LDA is useful when the variables are distributed normally and number of the categoric variables are too high. But the LDA does not work well for non-linear problems. Additionally, LDA needs validation and tends to be overfitting.

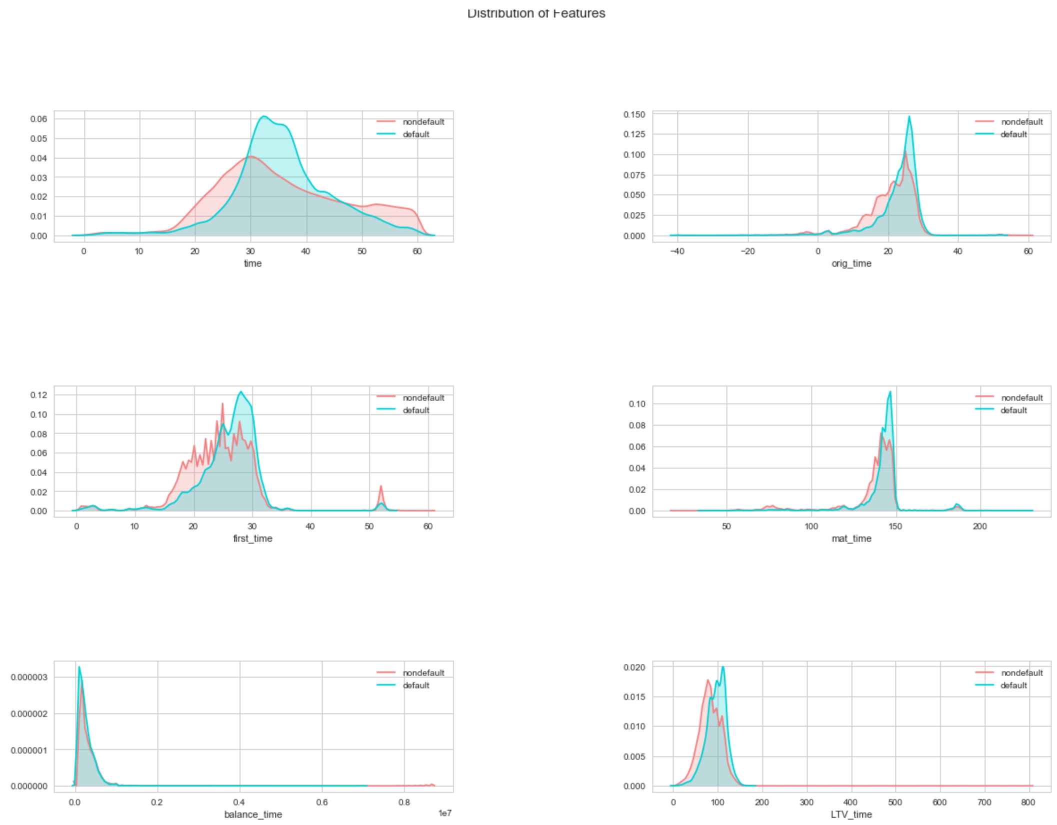
QDA is very similar with LDA but there is only a difference in probability function. While LDA is a linear function, QDA is a quadratic.

3. MODELLING

3.1 Exploratory Data Analysis

The mortgage dataset[2] consists of the observations for 50,000 residential U.S. mortgage borrowers over 60 periods. These loans may be originated before observation period and the start time and selected periods have not been defined in the dataset. The observations have been selected randomly from the portfolios underlying U.S. residential mortgage-backed securities (RMBS) securitization portfolios provided by International Financial Research. [3]

During the exploratory data analysis features were examined based on their distributions.



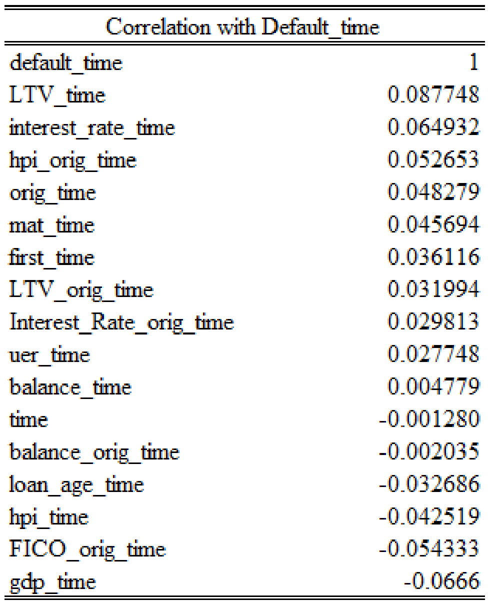
[2] The mortgage data can be uploaded from B. Baesens, D. Roesch, H. Scheule, Credit Risk Analytics: Measurement Techniques, Applications and Examples in SAS, Wiley, 2016. http://www.creditriskanalytics.net/datasets-private.html

[3] Source: www.internationalfinancialresearch.org

Figure-2: Examples for the distribution of some features

I observed the distribution of the dataset and defined some skewed features, but I did not transform all of them. During the data preprocessing I removed the outliers and abnormal values by using those distributions.

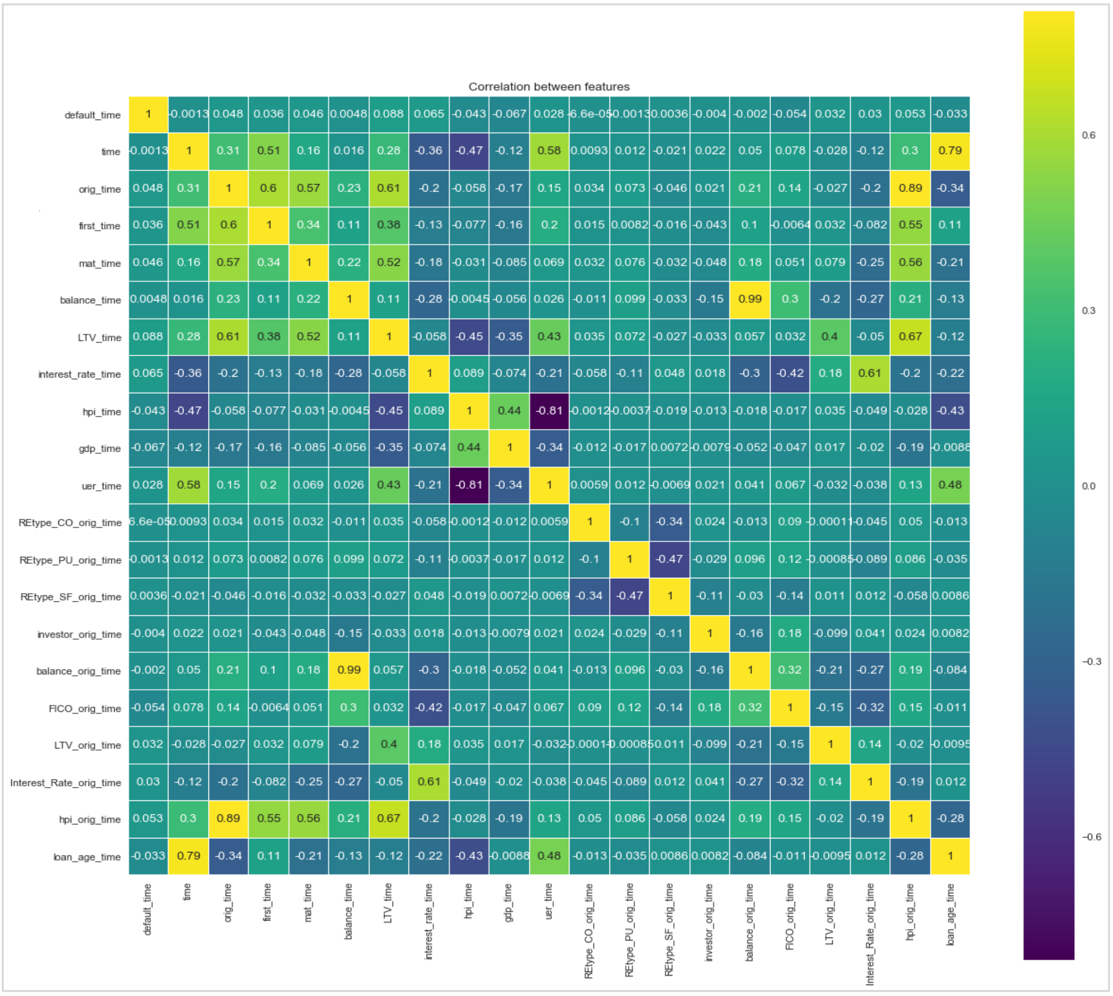
In order to detect the relationship of the features with the default ratio, correlation analysis was performed. As a result of the mentioned analysis, it was seen that there was no high correlation between features and default ratio (Figure-3).



Correlation with Default_time	
default_time	1
LTV_time	0.087748
interest_rate_time	0.064932
hpi_orig_time	0.052653
orig_time	0.048279
mat_time	0.045694
first_time	0.036116
LTV_orig_time	0.031994
Interest_Rate_orig_time	0.029813
uer_time	0.027748
balance_time	0.004779
time	-0.001280
balance_orig_time	-0.002035
loan_age_time	-0.032686
hpi_time	-0.042519
FICO_orig_time	-0.054333
gdp_time	-0.0666

Figure-3: Correlations between features and target

On the other hand, the high correlation between features was defined (Figure-4). It was seen that the features cover the ground the loan and economic outlook. This was not common for the default prediction models and this was a distinctive characteristic of my dataset. Because of this specificity of the dataset, the prediction models developed are much more dependent on the economic and loan information than customer information. Suitable systems and databases should be designed in order to embed these models in the real-business applications.



	default_time	time	orig_time	first_time	mat_time	balance_time	LTV_time	interest_rate_time	hpi_time	gdp_time	uer_time	REtype_CO_orig_time	REtype_PU_orig_time	REtype_SF_orig_time	investor_orig_time	balance_orig_time	FICO_orig_time	LTV_orig_time	Interest_Rate_orig_time	hpi_orig_time	loan_age_time
default_time	1	-0.0013	0.048	0.036	0.046	0.0048	0.088	0.065	-0.043	-0.067	0.028	-6.6e-05	-0.0013	0.0036	-0.004	-0.002	-0.054	0.032	0.03	0.053	-0.033
time	-0.0013	1	0.31	0.51	0.16	0.016	0.28	-0.36	-0.47	-0.12	0.58	0.0093	0.012	-0.021	0.022	0.05	0.078	-0.028	-0.12	0.3	0.79
orig_time	0.048	0.31	1	0.6	0.57	0.23	0.61	-0.2	-0.058	-0.17	0.15	0.034	0.073	-0.046	0.021	0.21	0.14	-0.027	-0.2	0.89	-0.34
first_time	0.036	0.51	0.6	1	0.34	0.11	0.38	-0.13	-0.077	-0.16	0.2	0.015	0.0082	-0.016	-0.043	0.1	-0.0064	0.032	-0.082	0.55	0.11
mat_time	0.046	0.16	0.57	0.34	1	0.22	0.52	-0.18	-0.031	-0.085	0.069	0.032	0.076	-0.032	-0.048	0.18	0.051	0.079	-0.25	0.56	-0.21
balance_time	0.0048	0.016	0.23	0.11	0.22	1	0.11	-0.28	-0.0045	-0.056	0.026	-0.011	0.099	-0.033	-0.15	0.99	0.3	-0.2	-0.27	0.21	-0.13
LTV_time	0.088	0.28	0.61	0.38	0.52	0.11	1	-0.058	-0.45	-0.35	0.43	0.035	0.072	-0.027	-0.033	0.057	0.032	0.4	-0.05	0.67	-0.12
interest_rate_time	0.065	-0.36	-0.2	-0.13	-0.18	-0.28	-0.058	1	0.089	-0.074	-0.21	-0.058	-0.11	0.048	0.018	-0.3	-0.42	0.18	0.61	-0.2	-0.22
hpi_time	-0.043	-0.47	-0.058	-0.077	-0.031	-0.0045	-0.45	0.089	1	0.44	-0.81	-0.0012	-0.0037	-0.019	-0.013	-0.018	-0.017	0.035	-0.049	-0.028	-0.43
gdp_time	-0.067	-0.12	-0.17	-0.16	-0.085	-0.056	-0.35	-0.074	0.44	1	-0.34	-0.012	-0.017	0.0072	-0.0079	-0.052	-0.047	0.017	-0.02	-0.19	-0.0088
uer_time	0.028	0.58	0.15	0.2	0.069	0.026	0.43	-0.21	-0.81	-0.34	1	0.0059	0.012	-0.0069	0.021	0.041	0.067	-0.032	-0.038	0.13	0.48
REtype_CO_orig_time	6.6e-05	0.0093	0.034	0.015	0.032	-0.011	0.035	-0.058	-0.0012	-0.012	0.0059	1	-0.1	-0.34	0.024	-0.013	0.09	-0.00011	-0.045	0.05	-0.013
REtype_PU_orig_time	-0.0013	0.012	0.073	0.0082	0.076	0.099	0.072	-0.11	-0.0037	-0.017	0.012	-0.1	1	-0.47	-0.029	0.096	0.12	-0.00085	-0.089	0.086	-0.035
REtype_SF_orig_time	0.0036	-0.021	-0.046	-0.016	-0.032	-0.033	-0.027	0.048	-0.019	0.0072	-0.0069	-0.34	-0.47	1	-0.11	-0.03	-0.14	0.011	0.012	-0.058	0.0086
investor_orig_time	-0.004	0.022	0.021	-0.043	-0.048	-0.15	-0.033	0.018	-0.013	-0.0079	0.021	0.024	-0.029	-0.11	1	-0.16	0.18	-0.099	0.041	0.024	0.0082
balance_orig_time	-0.002	0.05	0.21	0.1	0.18	0.99	0.057	-0.3	-0.018	-0.052	0.041	-0.013	0.096	-0.03	-0.16	1	0.32	-0.21	-0.27	0.19	-0.084
FICO_orig_time	-0.054	0.078	0.14	-0.0064	0.051	0.3	0.032	-0.42	-0.017	-0.047	0.067	0.09	0.12	-0.14	0.18	0.32	1	-0.15	-0.32	0.15	-0.011
LTV_orig_time	0.032	-0.028	-0.027	0.032	0.079	-0.2	0.4	0.18	0.035	0.017	-0.032	-0.0001	-0.00085	0.011	-0.099	-0.21	-0.15	1	0.14	-0.02	-0.0095
Interest_Rate_orig_time	0.03	-0.12	-0.2	-0.082	-0.25	-0.27	-0.05	0.61	-0.049	-0.02	-0.038	-0.045	-0.089	0.012	0.041	-0.27	-0.32	0.14	1	-0.19	0.012
hpi_orig_time	0.053	0.3	0.89	0.55	0.56	0.21	0.67	-0.2	-0.028	-0.19	0.13	0.05	0.086	-0.058	0.024	0.19	0.15	-0.02	-0.19	1	-0.28
loan_age_time	-0.033	0.79	-0.34	0.11	-0.21	-0.13	-0.12	-0.22	-0.43	-0.0088	0.48	-0.013	-0.035	0.0086	0.0082	-0.084	-0.011	-0.0095	0.012	-0.28	1

Figure-4: Correlations between features

Finally, a trend analysis was performed for the target variable (default_time). As it can be seen in the Figure-5 the default ratio reaches to a highest value between 30th and 40th months.

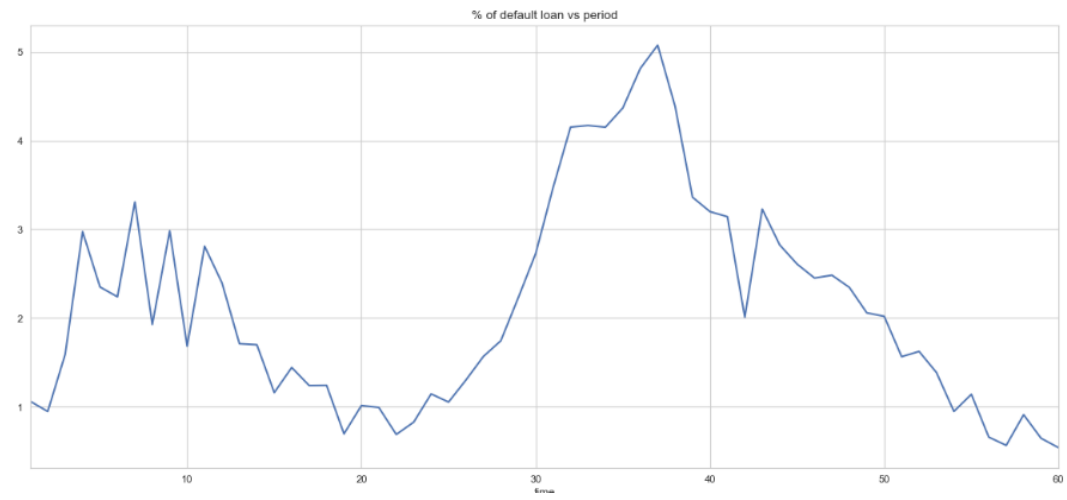


Figure-5: Trend analysis for the default ratio

3.2 Data Preprocessing

In our mortgage data set, the loans in the data are numbered consecutively (id = 1 to 50,000) and the observation periods (time = 1 to 60) are numbered consecutively from the first observation period. Loans may have originated prior to the first observation period and may carry negative values in such cases. Examples of loan-specific variables are the FICO score at origination and the loan-to-value (LTV) ratio at origination. The FICO mortgage score is a credit score with values between 300 and 850. LTV is the ratio of the outstanding loan amount to the collateral value, and banks traditionally extend loans in the region of 80 percent. The unemployment rate is time-varying and has the same value for all loans in a given observation period.

The main purpose of this phase is to analyse the dataset in order to reduce noise or incorrect data. To do this, firstly the existence of the missing value was examined and 270 missing values in LTV_time detected. LTV is calculated by the ratio of outstanding loan to the value of the house at that point in time. Thus, LTV can only be equal to zero when the customer pays for all the money she borrowed. In that case, a) the customer pays her all debt before the maturity, i.e. payoff_time is equal to 1 (or status_time is equal to 2) b) the customer pays her final debt at the maturity, i.e when maturity is equal to loan_age_time. There were 38 incorrect records not complying described conditions, namely for these records both LTV_time and balance_time are zero, yet neither status_time is 2 nor loan_age_time is equal to mat_time. These 38 incorrect records were dropped from the dataset.

Besides, there were negative values in the orig_time since loans may originate before the start of the observation period. In order to handle this situation, a new feature called loan_age_time was created by subtracting orig_time from time.

Additionally, the interest rates were examined and some 0 values are evaluated as remarkable. As it can be seen in the following figure mortgage interest rates have never been 0 in last 30 years. It was presumed that there was no information about the Interest_Rate when the mortgage was granted. Therefore it was decided to impute Interest_Rate_orig_time with the mode of the non-zero interest rate at the origin for each different origin time. It was also seen that from the figure below, the average interest rate for the last 30 years in the US has not been greater than 18%. Since this was the average interest rate, the records where interest_rate_time is 37.5 were accepted as outliers and dropped from the dataset.

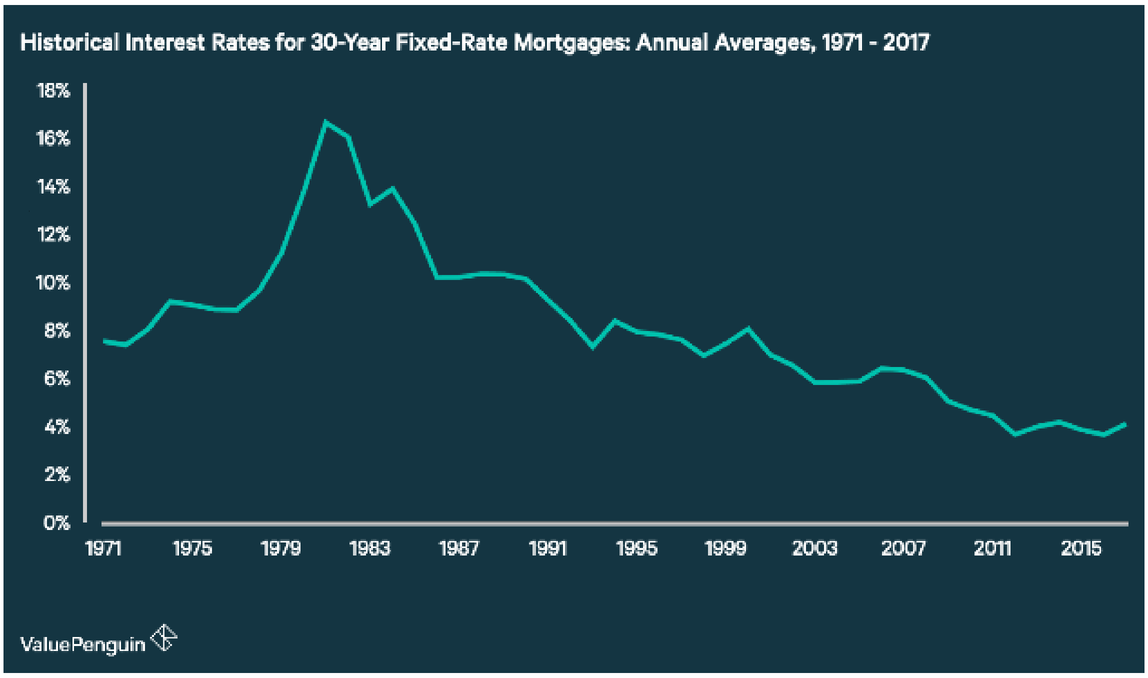


Figure-6: Historical interest rates[4]

3.3 Classification Models with Imbalanced Dataset

The dataset which I used for developing the models was imbalanced, namely, the ratio of the total number of defaults to the total number of non-defaults was too small. During the data preprocessing phase I detected this problem and executed the Light GBM model as a base model for the dataset to see the impact on the accuracy. As a result I obtained 0.9746 accuracy while recall was 0. With this result, I can say that Light GBM model predicted almost all the test data as non-default because of the high number of the non-default cases. In summary, even though the accuracy of the classification model is high for imbalanced dataset, the selected models cannot predict any of the defaults. In order to handle this imbalanced dataset problem, I applied several sampling methods. Additionally, I used recall, precision and F1 metrics as performance indicators with the accuracy.

3.3.1 Imbalanced Data Approaches

Imbalanced data is the non-equality problem when the total number of a feature (default) is much more lower than the total number of another feature (non-default). This problem might be seen in various cases related to fraud detection, anomaly detection, facial

[4]https://www.valuepenguin.com/mortgages/historical-mortgage-rates

recognition, default prediction etc. It is important to rearrange the dataset depending on the model’s target in case of imbalanced data. On the other hand, the developed models with an imbalanced dataset like Decision Tree and Logistic Regression might have biased result and tend to ignore minority class as predicting every outcome as majority class results very high accuracy. Moreover, model evaluation metrics such as accuracy cannot measure the model performance accurately. In order to avoid this, confusion matrix, precision, and recall metrics should be examined carefully during the model performance evaluation.

In order to deal with imbalanced dataset problems, some methods are used such as cost-sensitive approach and sampling approaches. In this study, the imbalanced data problem was solved by using sampling approaches. Because of this, in this section\ only the sampling approaches (under-sampling, over-sampling) were explained.

Under-sampling means removing some of the majority class instances thus the number of the minority is more close or equal to the majority class. Over-sampling means duplicating the minority classes till to defined level or to the majority.

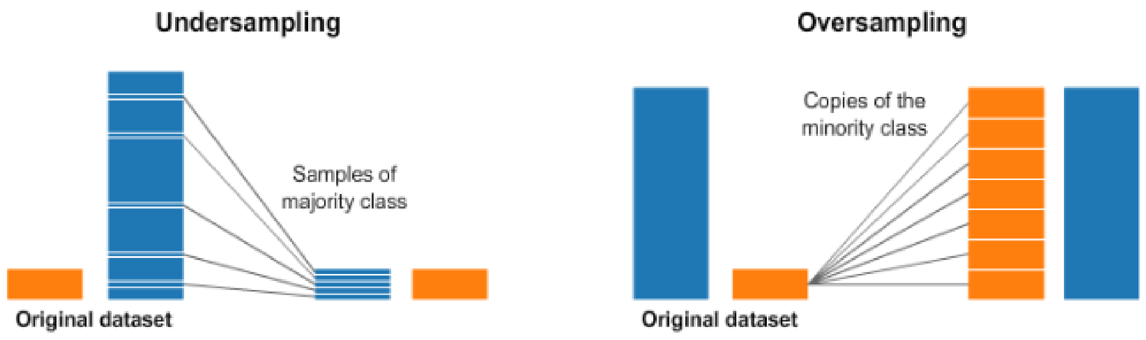


Figure-7: Under and over sampling illustration [5]

3.3.1.1 Removing Duplicates

I reduced the imbalance ratio and preserved a reasonable amount of information by removing the duplicates since in the dataset there were more than one record for each loan.

[5]https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets
https://www.svds.com/learning-imbalanced-classes/
https://towardsdatascience.com/dealing-with-imbalanced-classes-in-machine-learningd43d6fa19d2
https://www.researchgate.net/application.ClientValidation.html?origPath=%2F

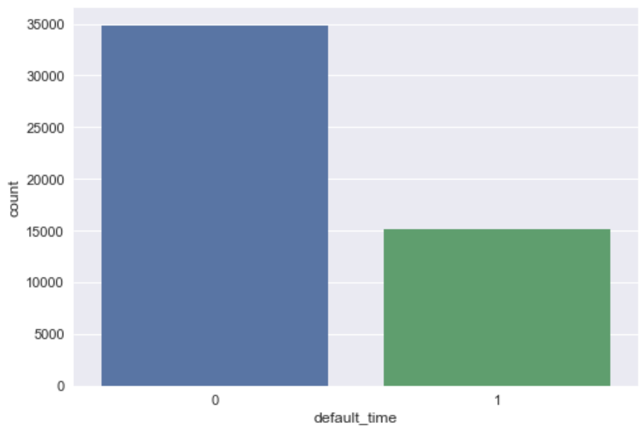


Figure-8: Distribution of the default cases after removing duplicates

By using the new dataset, I developed models with 18 features. For the comparison of models, I used to test F1 score because this score provides the balance between the precision and the recall. XGBoost algorithm ensured that the best result with the new dataset (test accuracy 0.828462; F1 score 0.704318).

Additionally, when I analyzed the computational speeds of the selected models, I noticed that the GaussianNB and the Decision Tree algorithms were a step ahead of the other.

	Classifier	Elapsed Time	Fit Time	Number of Features	Test Accuracy	Test F1	Test Precision	Test Recall
1	**XGBoost**	**40.540.163**	**14.266.215**	**18**	**0.828462**	**0.704318**	**0.734925**	**0.681619**
2	AdaBoost	124.813.394	42.996.092	18	0.821119	0.689307	0.727596	0.659593
3	Random Forrest	72.679.865	13.431.776	18	0.819798	0.686273	0.724697	0.660078
4	GaussianNB	15.198.386	0.047900	18	0.792926	0.677286	0.645723	0.724845
5	Quadratic Discriminant Analysis	16.180.057	0.082749	18	0.706347	0.635267	0.512007	0.845992
6	Extra Trees	56.858.702	5.563.478	18	0.805412	0.634921	0.738008	0.564425
7	Logistic Regression	17.178.282	0.946988	18	0.801149	0.630063	0.724455	0.561383
8	Linear Discriminant Analysis	16.002.195	0.143948	18	0.787002	0.612303	0.691661	0.557577
9	Decision Tree	15.952.746	0.618242	18	0.744284	0.588702	0.572666	0.606714
10	KNeighbors	20.023.298	0.162898	18	0.684475	0.368232	0.469273	0.307324

Table-1: A comparison or the models with removing duplicates

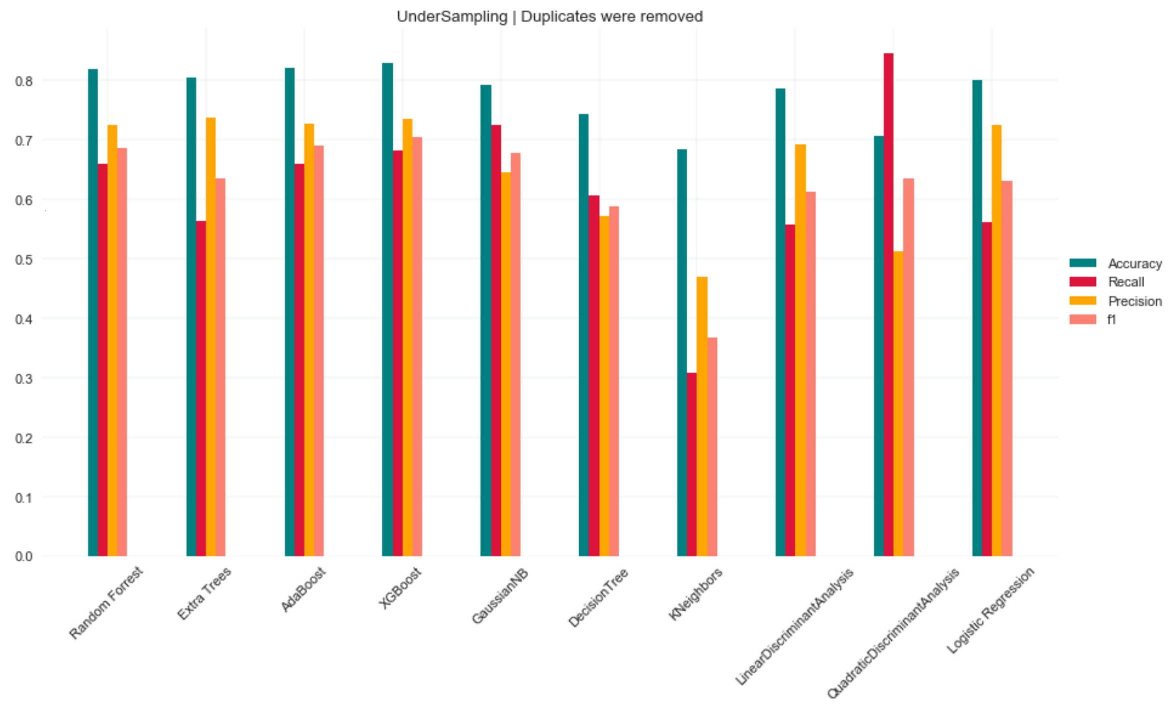


Figure-9: Performance metrics according to models

On the above (Figure-9), the performance metrics' results for the selected models can be seen. As I mentioned before, with XGBoost algorithm, all the metrics provided the better and optimal result for the dataset.

It is important note that removing observation is not a common method for machine learning algorithms since each observation has an informative value for the model. To do so, each approach and dataset should be examined and evaluated based on related performance metrics.

3.3.1.2 Under-Sampling

Under-sampling simply means that removing observations from majority class until the defined level. In my dataset, I removed some of the non-default observations until reaching to 1-1 ratio with the default observation.

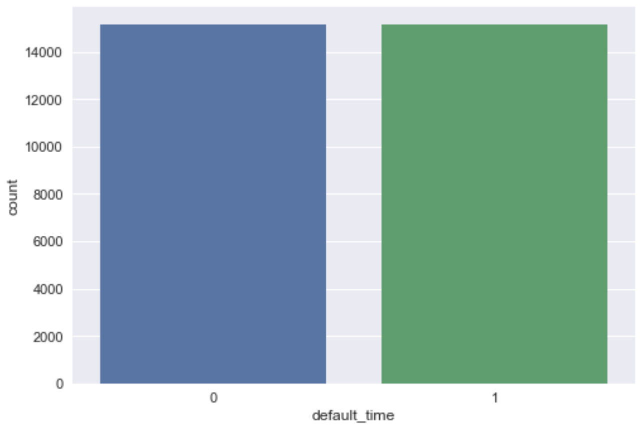
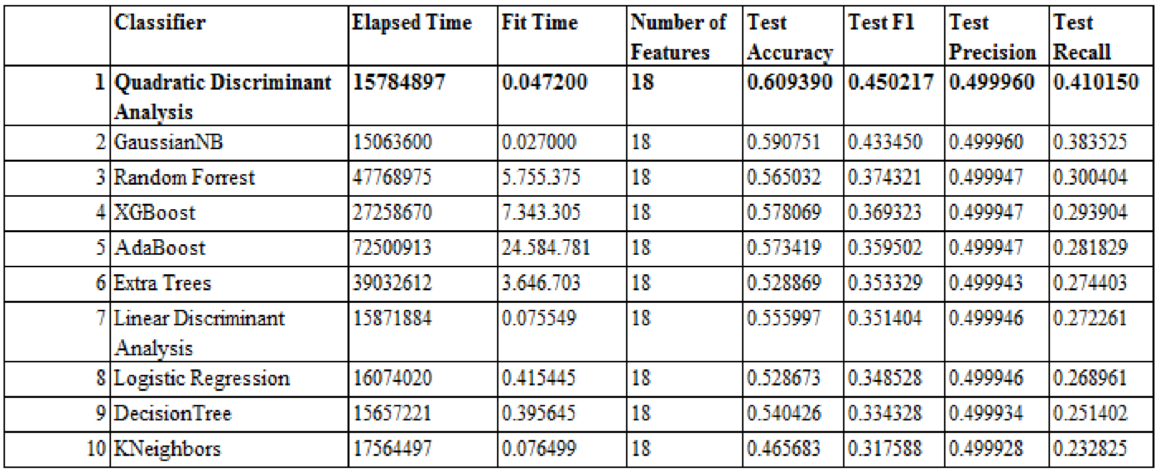


Figure-10: Distribution of the default cases with under-sampling

I developed models with 18 features similar with the previous approach. When I compared the models by using test F1 score, I saw that Quadratic Discriminant Analysist (QDA) algorithm ensured that the best result (test accuracy 0.609390, F1 score 0.450217). In general, if covariances are very distinct for the different classes, QDA probably provides good results.

Additionally, when I analyzed the computational speeds of the selected models, I noticed that all the models have an advantage in terms of computational speed, except first four models.



	Classifier	Elapsed Time	Fit Time	Number of Features	Test Accuracy	Test F1	Test Precision	Test Recall
1	**Quadratic Discriminant Analysis**	**15784897**	**0.047200**	**18**	**0.609390**	**0.450217**	**0.499960**	**0.410150**
2	GaussianNB	15063600	0.027000	18	0.590751	0.433450	0.499960	0.383525
3	Random Forrest	47768975	5.755.375	18	0.565032	0.374321	0.499947	0.300404
4	XGBoost	27258670	7.343.305	18	0.578069	0.369323	0.499947	0.293904
5	AdaBoost	72500913	24.584.781	18	0.573419	0.359502	0.499947	0.281829
6	Extra Trees	39032612	3.646.703	18	0.528869	0.353329	0.499943	0.274403
7	Linear Discriminant Analysis	15871884	0.075549	18	0.555997	0.351404	0.499946	0.272261
8	Logistic Regression	16074020	0.415445	18	0.528673	0.348528	0.499946	0.268961
9	DecisionTree	15657221	0.395645	18	0.540426	0.334328	0.499934	0.251402
10	KNeighbors	17564497	0.076499	18	0.465683	0.317588	0.499928	0.232825

Table-2: A comparison for the models with under-sampling dataset

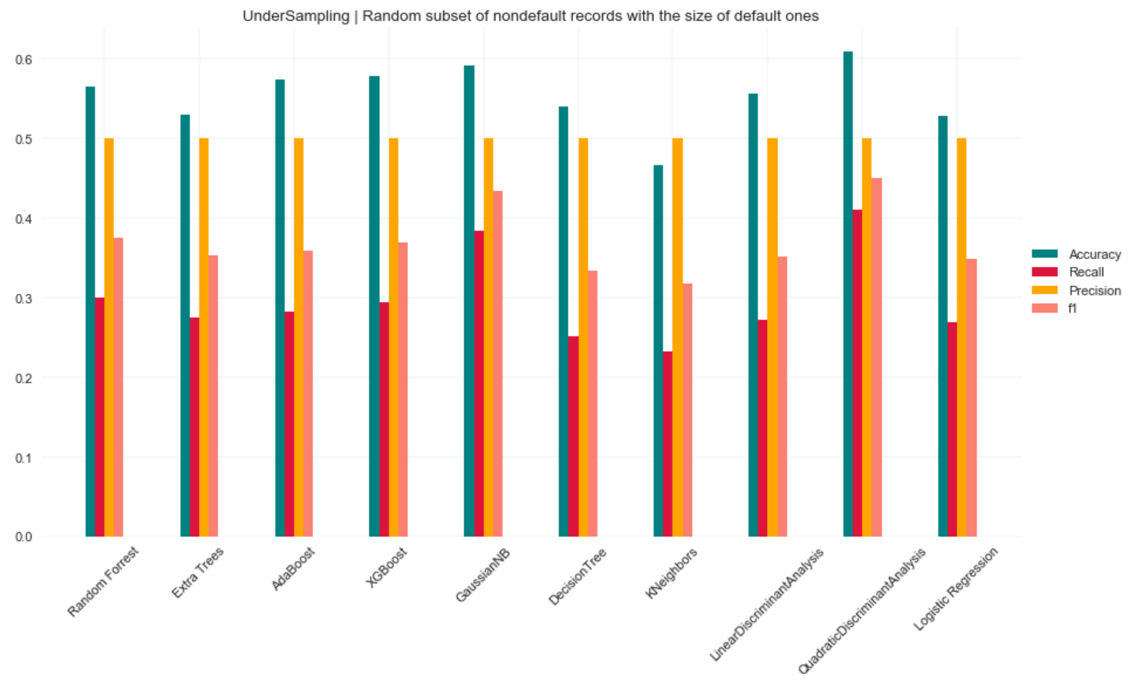


Figure-11: Performance metrics according to models

On the above (Figure-11), the performance metrics' results for the selected models can be seen. When I compare the results for precision and recall, I can say easily the models cannot predict default cases properly. (Steyerberg, E.W.; Vickers, A. J.; Cook, N. R.; Gerds, T.; Gonen, M.; Obuchowski, N.; Pencina, M.J.; Kattan, W. M., 2010).

3.3.1.3 Over-Sampling

Oversampling means that adding additional data to the minority class. There are several approaches for over-sampling such as duplicating samples in the minority class or SMOTE (Synthetic Minority Oversampling Technique).

In SMOTE, minority class is over-sampled by taking each minority class sample and introducing synthetic examples along the line segments joining any/ all of the k minority class nearest neighbors. Depending upon the amount of over-sampling required, neighbors from the k nearest neighbors are randomly chosen[6]. It is important note that splitting the dataset as train and test have to performed before SMOTE and this technique only has to be applied on train dataset. On the other hand, the test dataset cannot reflect the actual situation for your problem/dataset[7].

[6] SMOTE: Synthetic Minority Over-sampling Technique, Journal of Artificial Intelligence Research 2002, pp.321–357
[7]https://beckernick.github.io/oversampling-modeling/

In my dataset, I added some of the non-default observations until reaching to 1-1 ratio with the default observation.

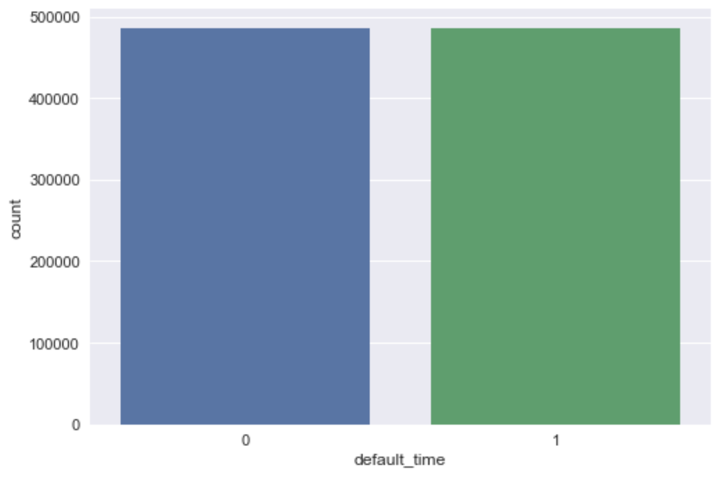
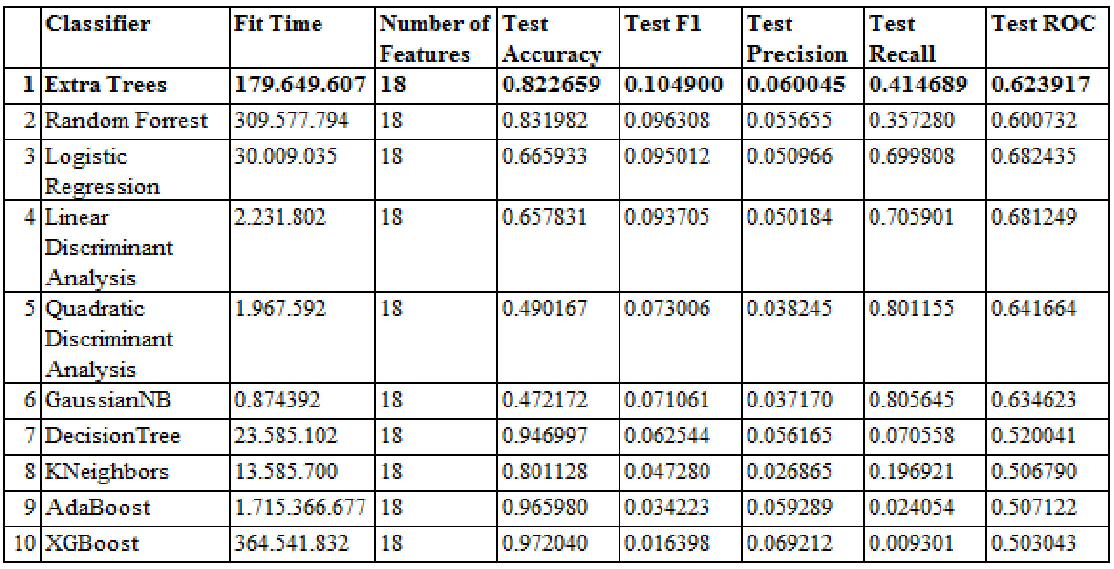


Figure-12: Distribution of the default cases with over-sampling

I developed models with 18 features similar to the previous approaches. When I compared the models' results, even if there were acceptable accuracies and ROC values, F1 score, precision and recall metrics were not in acceptable level. Because of this, I did not find the SMOTE technique useful for my dataset.



	Classifier	Fit Time	Number of Features	Test Accuracy	Test F1	Test Precision	Test Recall	Test ROC
1	**Extra Trees**	**179.649.607**	**18**	**0.822659**	**0.104900**	**0.060045**	**0.414689**	**0.623917**
2	Random Forrest	309.577.794	18	0.831982	0.096308	0.055655	0.357280	0.600732
3	Logistic Regression	30.009.035	18	0.665933	0.095012	0.050966	0.699808	0.682435
4	Linear Discriminant Analysis	2.231.802	18	0.657831	0.093705	0.050184	0.705901	0.681249
5	Quadratic Discriminant Analysis	1.967.592	18	0.490167	0.073006	0.038245	0.801155	0.641664
6	GaussianNB	0.874392	18	0.472172	0.071061	0.037170	0.805645	0.634623
7	DecisionTree	23.585.102	18	0.946997	0.062544	0.056165	0.070558	0.520041
8	KNeighbors	13.585.700	18	0.801128	0.047280	0.026865	0.196921	0.506790
9	AdaBoost	1.715.366.677	18	0.965980	0.034223	0.059289	0.024054	0.507122
10	XGBoost	364.541.832	18	0.972040	0.016398	0.069212	0.009301	0.503043

Table-3: A comparison for the models with SMOTE

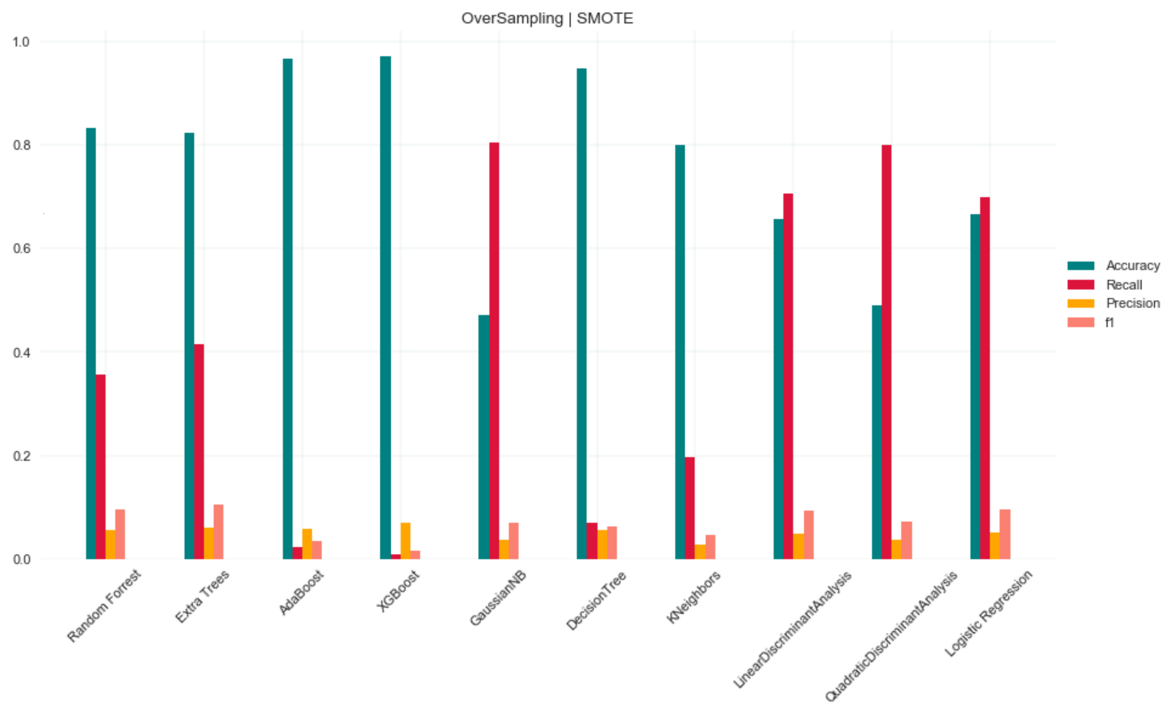


Figure-13: Performance metrics according to models

In the above (Figure-13), the abnormal values for accuracy, recall, precision and F1 can be seen. As I mentioned before, SMOTE did not provide any improvement for my dataset. In order to handle this problem, using other approaches or collecting additional observation are the methods that should be considered.

3.4 Evaluation of Results

In this study, I focused on developing prediction models for mortgage loans. I preferred to use classification methods such as Random Forest, Extra Trees, XGBoost etc. since the target outputs had to be binary. In the beginning, I developed a base model by using Light GBM model and I obtained high accuracy (0.9746). I evaluated it as considerable and dived deeply into explanatory data analysis. Even if the dataset relatively was in a clean format, I faced with the imbalanced dataset problem. In order to solve this problem, several methods are recommended such as removing duplicates, under-sampling, over-sampling, SMOTE techniques and weighted sampling in the literature.

Firstly, I removed the duplicates from the dataset and obtained the best result with the XGBoost modeling algorithm. Although I lost some of the information related to loan and economic outlook with this approach, depending on my dataset, results still were satisfactory (test accuracy: 0.828462; F1 score 0.704318). As aligned with the literature, XGBoost performed a better fit than other models and capture high-order interactions without breaking down. It is worth mentioning that when I develop the models with the several methods; all the parameters were also tuned manually.

Then, I used under-sampling, SMOTE and weighted sampling methods but the results were not acceptable level since F1 score, precision and recall metrics did not explain the problem I focused. There were not any balance between these metrics and they did not predict the default cases concretely.

As a result, even if I obtained the better results with the XGBoost algorithm by removing duplicates, I cannot express that the method selected is an absolute correct. It is important that in the machine learning algorithms; there is no unique correct method for developing a model. In order to select a most suitable model and methods, different approaches should be tried considering dataset's specificities.

For the future study, weighted sampling should be considered. In the literature, this method has been discussed by Kashtan, Itzkovitz, Milo, & Alon (Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs, 2004). Some algorithms such as random forest, extra tree, XGBOOST etc. have their parameters to give weight the selected features in dataset. By using these algorithms, increasing of the models' predictive powers should be tried. Additionally, ensemble sampling methods also should be considered for imbalanced dataset problems.

4. REFERENCES

4.1 REFERENCES

Akindaini, B. (2017). Machine Learning Applications in Mortgage. *University of Tampere Faculty of Natural Sciences Master's Degree Programme in Computational Big Data Analytics* .

Bajari, P., Chu, C. S., & Park, M. (2008). An Empirical Model of Subprime Mortgage Default From 2000 to 2007. *NBER Working Paper No. 14625*.

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* , pp. 785-794.

Deng, Y., Quigley, M. J., & Van Order, R. (2003). Mortgage Terminations, Heterogeneity and the Exercise of Mortgage Options. *Econometrica*, pp. 275-307.

Fitzpatrick, T. (2016). An empirical comparison of classification algorithms for mortgage default prediction: evidence from a distressed mortgage market. *European Journal of Operational Research*, pp. 427-439.

Furstenberg, G. M. (1970). The Investment Quality of Home Mortgages. *The Journal of Risk and Insurance*, pp. 437-445.

Gerardi, K., & Willen, P. (2009). Subprime Mortgages, Foreclosures, and Urban Neighborhoods. *The B.E. Journal of Economic Analysis & Policy*.

Han, H., WY, W., & Mao, B. (2005). Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. *Advances in Intelligent Computing. ICIC 2005. Lecture Notes in Computer Science*, pp. 878-887.

Kashtan, N., Itzkovitz, S., Milo, R., & Alon, U. (2004). Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics*, pp. 1746–1758.

Khandani, A. E. (2010). Consumer credit-risk models via machine-learning algorithms. *Journal of Banking & Finance*, pp. 2767-2787.

Lyn, C. (2000). A survey of credit and behavioural scoring: forecasting financial risk of lending to consumers. *International Journal of Forecasting*, pp. 149-172.

Morton, T. G. (1975). *A Discriminant Function Analysis of Residential Mortgage Delinquency and Foreclosure.*

Pivo, G. (2014). he Effect of Sustainability Features on Mortgage Default Prediction and Risk in Multifamily Rental Housing. *Journal of Sustainable Real Estate*, pp. 149-170.

Provost, F. (2000). Machine Learning from Imbalanced Data Sets. *AAAI Technical Report WS-00-05*.

Quercia, R., & Stegman, A. (1992). Residential Mortgage Default: A Review of the Literature . *Journal of Housing Research*, pp. 341-379.

Rosenblatt, G., & Crawford, W. E. (1999). Differences in the Cost of Mortgage Credit Implications for Discrimination. *The Journal of Real Estate Finance and Economics*, pp. 147–159.

Steyerberg, E.W.; Vickers, A. J.; Cook, N. R.; Gerds, T.; Gonen, M.; Obuchowski, N.; Pencina, M.J.; Kattan, W. M. (2010). Assessing the performance of prediction models: a framework for some traditional and novel measures. *Epidemiology*, pp. 128-138.

Vandell, K. D. (1996). Handing Over the Keys: A Perspective on Mortgage Default Research. *Real Estate Economics* , pp. 211-246.

Webb, B. G. (1982). Borrower Risk under Alternative Mortgage Instruments. *The Journal of Finance*, pp 169-183.

West, D. (2000). Neural network credit scoring models. *Computers & Operations Research*, pp. 1131-1152.

Dealing with Imbalanced Classes in Machine Learning; "https://towardsdatascience.com/dealing-with-imbalanced-classes-in-machine-learningd43d6fa19d2"

Dealing with Imbalanced Data: Under-sampling, Over-sampling and Proper Cross Validation, "https://www.marcoaltini.com/blog/dealing-with-imbalanced-data-undersampling-oversampling-and-proper-cross-validation"

The Right Way to Oversample in Predictive Modeling, "https://beckernick.github.io/oversampling-modeling/ "

How to Handle Imbalanced Classification Problems in Machine Learning? "https://www.analyticsvidhya.com/blog/2017/03/imbalanced-classification-problem/"

Unbalanced Data and Cross-validation,
“https://www.kaggle.com/questions-and-answers/27589”

Evaluation Measures for Models Assessment over Imbalanced Data Sets, “https://eva.fing.edu.uy/pluginfile.php/69453/mod_resource/content/1/7633-10048-1-PB.pdf”

Resampling Strategies for Imbalanced Datasets,
“https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets”

SMOTE: Synthetic Minority Over-Sampling Technique,
“https://jair.org/index.php/jair/article/view/10302/24590”

Classification Accuracy is not Enough: More Performance Measures You Can Use, “https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/”

Thoughts on Machine Learning – Dealing with Skewed Classes,
“https://florianhartl.com/thoughts-on-machine-learning-dealing-with-skewed-classes.html”

Tree- Based Algorithms,
“https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/”,
“https://lagunita.stanford.edu/c4x/HumanitiesScience/StatLearning/asset/trees.pdf”
“https://sadanand-singh.github.io/posts/treebasedmodels/”

Explanation of the Decision Tree,
“https://infocenter.informationbuilders.com/wf80/index.jsp?topic=%2Fpubdocs%2FRStat16%2Fsource%2Ftopic47.htm”

Boosting Algorithms,
“https://www.analyticsvidhya.com/blog/2015/11/quick-introduction-boosting-algorithms-machine-learning/”

Random Forest Algorithm,
https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm,
http://homes.sice.indiana.edu/classes/spring2018/info/i529-yye/lectures/DecisionTree.pdf,

https://stats.stackexchange.com/questions/175523/difference-between-random-forest-and-extremely-randomized-trees

K-Neighbors Algorithm,

“http://scikit-learn.org/stable/modules/neighbors.html#neighbors”

Logistic Regression,

“https://machinelearning-blog.com/2018/04/23/logistic-regression-101/”

“https://www.hackingnote.com/en/machine-learning/algorithms-pros-and-cons/”

“https://sebastianraschka.com/Articles/2014_python_lda.html”

“http://scikit-learn.org/stable/modules/lda_qda.html#lda-qda”

APPENDIX A

Github link: https://github.com/itezgiden/capstone/blob/master/Capstone.ipynb

Jupyter Notebook Codes: “DEFAULT PREDICTION MODELS FOR MORTGAGE LOANS”

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
# imported for drawing graphs
import seaborn as sns;sns.set(style="whitegrid",color_codes=True) #data visualization and tune set params
import matplotlib.pyplot as plt;plt.figure(figsize=(15,15)) #data plotting/visualization
import matplotlib as mpl
from scipy import stats

# Imported in order to handle skewness
from scipy.stats import norm, skew #for some statistics
from scipy.stats import boxcox
from scipy.special import boxcox1p

#imported in order to convert ordinal categorical number into number
from sklearn.preprocessing import LabelEncoder

# Machine Learning Algorithms
# Classifiers
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost.sklearn import XGBClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis,
QuadraticDiscriminantAnalysis
import lightgbm
from sklearn import tree
# Regressors
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

# imported to calculate cross validation score
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
from sklearn.model_selection import GridSearchCV
from sklearn import cross_validation

# imported to calculate accuracy
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import roc_auc_score

from sklearn.metrics import make_scorer
from sklearn.metrics import confusion_matrix

#imported to split data for test and train
from sklearn.model_selection import train_test_split

# imported to handle imbalanced data
from imblearn.over_sampling import SMOTE
#imported to calculate execution time of each model.
```

```
import timeit
%matplotlib inline
#Loading Dataset
rawdata = pd.read_csv("C:\\capstone\\mortgage.csv")
print ("raw data has {0} rows and {1} columns".format(rawdata.shape[0],rawdata.shape[1]))

# set seed for reproducibility
np.random.seed(0)

rawdata.head()
for c in rawdata.columns:
    print(c,":",rawdata[c].dtypes)
rawdata.describe().T
rawdata.isnull().sum()
rawdata['loan_age_time'] = rawdata['time'] - rawdata['orig_time']
rawdata[['loan_age_time','time','orig_time'] ].head()
rawdata[(rawdata['LTV_time'].isnull())].groupby(['status_time']).size()

mortgage = rawdata[~(rawdata['LTV_time'].isnull())]
mortgage[
            (
                (mortgage['balance_time']==0) &
                (mortgage['status_time']!=2)
            )
        ].shape

incorrect_LTV = mortgage[
            (mortgage['LTV_time']==0) &
            (mortgage['status_time']!=2)
        ][['id','balance_time','loan_age_time','mat_time',
           'payoff_time','LTV_time','status_time']]
display( incorrect_LTV,
    print('{0} of LTV in the dataset is wrong'.format(incorrect_LTV.shape[0]))  )
mortgage = mortgage[ ~(
                (mortgage['LTV_time']==0) &
                (mortgage['status_time']!=2)) ]

mortgage[(
                (mortgage['balance_time']==0) &
                (mortgage['status_time']!=2) ) ]

orig_interest_rate_removed = []
orig_interest_rate0 = mortgage[
                                (mortgage['Interest_Rate_orig_time'] == 0)
                              ]['orig_time'].unique()

for i in range(0,len(orig_interest_rate0)):
    tmp = [x for x in mortgage[
mortgage['orig_time']==orig_interest_rate0[i]]['Interest_Rate_orig_time'] if x !=0]
    if not tmp:
        orig_interest_rate_removed.append(orig_interest_rate0[i])

orig_interest_rate_imputed =[x for x in orig_interest_rate0
                      if x not in orig_interest_rate_removed]

print(  'when orig_time = {},all orig_interest_rate variables are
zero.'.format(orig_interest_rate_removed))
print('\n')
print(  'when orig_time = {}, \nwe have reasonable amount of data \
about orig_interest_rate at that time and can impute orig_interest_rate=0 \
with the mode/mean of non-zero orig_interest_rate.'.format(orig_interest_rate_imputed))
```

```
# Create a figure space matrix consisting of 3 columns and 2 rows
# Here is a useful template to use for working with subplots.
####################################################################
fig, ax = plt.subplots(figsize=(15,40), ncols=4, nrows=14)
left   =  0.125  # the left side of the subplots of the figure
right  =  0.9    # the right side of the subplots of the figure
bottom =  0.1    # the bottom of the subplots of the figure
top    =  0.9    # the top of the subplots of the figure
wspace =  .5     # the amount of width reserved for blank space between subplots
hspace =  1.1    # the amount of height reserved for white space between subplots

# This function actually adjusts the sub plots using the above paramters
plt.subplots_adjust(
    left    =  left,
    bottom  =  bottom,
    right   =  right,
    top     =  top,
    wspace  =  wspace,
    hspace  =  hspace)

# The amount of space above titles
y_title_margin = 0.8
plt.suptitle("Interest Rate Distribution at Origin (when there is at least one Interest
Rate=0)",
             y = 0.94, fontsize=15)
for i in range(0,len(orig_interest_rate_imputed)):
    ax[np.int(i/4)][np.int(i%4)].set_title("orig_time={0}".format(
                                            orig_interest_rate_imputed[i]),
                                 y = y_title_margin)
    tmp = mortgage[
mortgage['orig_time']==orig_interest_rate_imputed[i]]['Interest_Rate_orig_time']
    sns.distplot(tmp, ax=ax[np.int(i/4)][np.int(i%4)] )
```

```
display(mortgage[ mortgage['orig_time']==34].groupby('Interest_Rate_orig_time').size())
orig_interest_rate_removed.append(34)
orig_interest_rate_imputed = [x for x in orig_interest_rate_imputed if x !=34]

print(orig_interest_rate_removed)
print(orig_interest_rate_imputed)

for i in orig_interest_rate_removed:
    display(
        mortgage[ mortgage['orig_time']==i][['orig_time','Interest_Rate_orig_time']]
    )
    #mortgage = mortgage[ ~(mortgage['orig_time']==i)]

mortgage.loc[ mortgage['orig_time']==34,
             ['Interest_Rate_orig_time']] = mortgage[
mortgage['time']==34]['interest_rate_time'].mode().values[0]

mortgage.loc[ mortgage['orig_time']==38,
             ['Interest_Rate_orig_time']] = mortgage[
mortgage['time']==38]['interest_rate_time'].mode().values[0]

mortgage.loc[ mortgage['orig_time']==39,
             ['Interest_Rate_orig_time']] = mortgage[
mortgage['time']==39]['interest_rate_time'].mode().values[0]

mortgage= mortgage[~(mortgage['orig_time']==-35)]

display(
    mortgage[ mortgage['orig_time']==-35][['orig_time','Interest_Rate_orig_time']],
    mortgage[ mortgage['orig_time']==34][['orig_time','Interest_Rate_orig_time']],
    mortgage[ mortgage['orig_time']==38][['orig_time','Interest_Rate_orig_time']],
    mortgage[ mortgage['orig_time']==39][['orig_time','Interest_Rate_orig_time']],)
```

```
for i in orig_interest_rate_imputed:
    tmp = [x for x in mortgage[ mortgage['orig_time']==i]['Interest_Rate_orig_time'] if
x!=0.0 ]
    impute_value = pd.Series(tmp).mode().values[0]
    mortgage.loc[ (mortgage['orig_time']==i) &
(mortgage['Interest_Rate_orig_time']==0),['Interest_Rate_orig_time'] ] = impute_value
    #print (i,'::',impute_value)
    #print (i,'::',impute_value)

mortgage[(mortgage['Interest_Rate_orig_time'] == 0)]['orig_time'].unique()

display(mortgage[(mortgage['interest_rate_time'] == 0) ])
sns.distplot(mortgage[(mortgage['time'] == 26) ]['interest_rate_time'] )

mortgage.loc[ (mortgage['time']==26) & (mortgage['interest_rate_time']==0),
            ['interest_rate_time']] =
mortgage[mortgage['time']==26]['interest_rate_time'].mode().values[0]

mortgage.loc[ (mortgage['time']==30) & (mortgage['interest_rate_time']==0),
            ['interest_rate_time']] =
mortgage[mortgage['time']==30]['interest_rate_time'].mode().values[0]

mortgage.loc[ (mortgage['time']==31) & (mortgage['interest_rate_time']==0),
            ['interest_rate_time']] =
mortgage[mortgage['time']==31]['interest_rate_time'].mode().values[0]
display(mortgage[(mortgage['interest_rate_time'] == 0) ])

mortgage.status_time.unique()

sns.countplot("default_time",data=mortgage)

%matplotlib inline
```

```
plt.figure(figsize=(18,8), dpi=80, facecolor='w', edgecolor='k')

prt_of_default = []
for t in np.sort(mortgage['time'].unique()):
   prt = len(mortgage[(mortgage['time']==t) & (mortgage['status_time']==1)]
)*100/len(mortgage[mortgage['time']==t])
   prt_of_default.append({'time':t,'percentage_of_default':prt })

prt_of_defaultDF = pd.DataFrame(prt_of_default)
prt_of_defaultDF.index = prt_of_defaultDF['time']
prt_of_defaultDF['percentage_of_default'].plot(title='% of default loan vs
period',grid=True)

x = np.sort(mortgage['FICO_orig_time'])
y = x.cumsum()/x.sum()
plt.plot(x,y,marker='.',linestyle='none')

x = np.sort(mortgage['LTV_orig_time'])
y = x.cumsum()/x.sum()
plt.plot(x,y,marker='.',linestyle='none')

import scipy.stats as stats
import pylab
measurements = mortgage['FICO_orig_time']
stats.probplot(measurements, dist="norm", plot=pylab)
pylab.show()

measurements = mortgage['LTV_orig_time']
stats.probplot(measurements, dist="norm", plot=pylab)
pylab.show()
```

```
fig, ax = plt.subplots(figsize=(15,10), ncols=2, nrows=2)

left   =  0.125  # the left side of the subplots of the figure
right  =  0.9    # the right side of the subplots of the figure
bottom =  0.1    # the bottom of the subplots of the figure
top    =  0.9    # the top of the subplots of the figure
wspace =  .5     # the amount of width reserved for blank space between subplots
hspace =  1.1    # the amount of height reserved for white space between subplots

# This function actually adjusts the sub plots using the above paramters
plt.subplots_adjust(
    left    =  left,
    bottom  =  bottom,
    right   =  right,
    top     =  top,
    wspace  =  wspace,
    hspace  =  hspace
)

# The amount of space above titles
y_title_margin = 1

plt.suptitle("Distribution of Categorical Variables over Default",
             y = 1, fontsize=15)

categoricalColumns = ['REtype_CO_orig_time', 'REtype_PU_orig_time',
'REtype_SF_orig_time',
                      'investor_orig_time']

categoricalColumnsDict = {'REtype_CO_orig_time':'condominium',
                          'REtype_PU_orig_time':'planned urban development',
                          'REtype_SF_orig_time':'Single-family home',
                  'investor_orig_time':'Investor borrower'
                 }

for i in range(0,len(categoricalColumns)):
    ax[np.int(i/2)][np.int(i%2)].set_title("Dist. of {0}".format(
                                          categoricalColumns[i]),
                             y = y_title_margin)
    # Set all labels on the row axis of subplots for bathroom data to "bathrooms"
    ax[np.int(i/2)][np.int(i%2)].set_ylabel("Count")
    ax[np.int(i/2)][np.int(i%2)].set_xlabel("Default")
    b = sns.countplot(x = 'default_time',hue = categoricalColumns[i], data =
mortgage,ax=ax[np.int(i/2)][np.int(i%2)])
    b.legend(["Otherwise",categoricalColumnsDict[categoricalColumns[i]]],fontsize = 11)

excluded = ['id','default_time','payoff_time','status_time']
categoricalColumns = ['REtype_CO_orig_time', 'REtype_PU_orig_time',
'REtype_SF_orig_time', 'investor_orig_time']
included = []
for c in mortgage.columns:
    if c not in excluded and c not in categoricalColumns:
        included.append(c)

# Here is a useful template to use for working with subplots.
######################################################################
fig, ax = plt.subplots(figsize=(20,40), ncols=2, nrows=8)

left   =  0.125  # the left side of the subplots of the figure
right  =  0.9    # the right side of the subplots of the figure
bottom =  0.1    # the bottom of the subplots of the figure
top    =  0.9    # the top of the subplots of the figure
wspace =  .5     # the amount of width reserved for blank space between subplots
hspace =  1.1    # the amount of height reserved for white space between subplots
```

```
# This function actually adjusts the sub plots using the above paramters
plt.subplots_adjust(
    left    =  left,
    bottom  =  bottom,
    right   =  right,
    top     =  top,
    wspace  =  wspace,
    hspace  =  hspace)

# The amount of space above titles
y_title_margin = 1

plt.suptitle("Boxplot of Numerical Variables",
             y = 0.94, fontsize=15)

for i in range(0,len(included)):
    ax[np.int(i/2)][np.int(i%2)].set_ylabel(included[i])
    ax[np.int(i/2)][np.int(i%2)].set_xlabel("Default")
    b=sns.factorplot(x = "default_time", y = included[i],data = mortgage,
kind="box",ax=ax[np.int(i/2)][np.int(i%2)])

    # According to https://stackoverflow.com/questions/33925494/seaborn-produces-
separate-figures-in-subplots
    # factorplot creates a new FacetGrid instance (which in turn creates its own figure), on
which it will
    # apply a plotting function (pointplot by default). Because of this, If I did not close
factorplot with
    # following line, it also creates empty facetgrid
    plt.close(b.fig)
```

```
fig, ax = plt.subplots(figsize=(15,10))
b=sns.factorplot(x = "default_time", y = 'interest_rate_time',data = mortgage,
kind="box",ax=ax)
plt.close(b.fig)

display('There are {0} outliers where interest_rate_time is equal to 37.5'.format(
        mortgage[mortgage['interest_rate_time']==37.5]['interest_rate_time'].count())
    )

mortgage = mortgage[mortgage['interest_rate_time']!=37.5]

fig, ax = plt.subplots(figsize=(15,10))
b=sns.factorplot(x = "default_time", y = 'interest_rate_time',data = mortgage,
kind="box",ax=ax)
plt.close(b.fig)

excluded = ['id','default_time','payoff_time','status_time']
#categoricalColumns = ['REtype_CO_orig_time', 'REtype_PU_orig_time',
'REtype_SF_orig_time', 'investor_orig_time']
included = []
for c in mortgage.columns:
  if c not in excluded and c not in categoricalColumns:
    included.append(c)

# Here is a useful template to use for working with subplots.
#
######################################################################
fig, ax = plt.subplots(figsize=(20,40), ncols=2, nrows=8)

left   =  0.125  # the left side of the subplots of the figure
right  =  0.9    # the right side of the subplots of the figure
bottom =  0.1    # the bottom of the subplots of the figure
```

```
top    =  0.9    # the top of the subplots of the figure
wspace =  .5     # the amount of width reserved for blank space between subplots
hspace =  1.1    # the amount of height reserved for white space between subplots

# This function actually adjusts the sub plots using the above paramters
plt.subplots_adjust(
    left    =  left,
    bottom  =  bottom,
    right   =  right,
    top     =  top,
    wspace  =  wspace,
    hspace  =  hspace)

# The amount of space above titles
y_title_margin = 1

plt.suptitle("Distribution of Features",
             y = 0.94, fontsize=15)

for i in range(0,len(included)):
    ax[np.int(i/2)][np.int(i%2)].set_ylabel("")
    ax[np.int(i/2)][np.int(i%2)].set_xlabel(included[i])
    k1=sns.kdeplot(mortgage[mortgage.default_time == 0][included[i]],
                color="lightcoral", shade=True,
                label='nondefault',ax=ax[np.int(i/2)][np.int(i%2)])
    k2=sns.kdeplot(mortgage[mortgage.default_time == 1][included[i]],
                color="darkturquoise", shade=True,
                label='default',ax=ax[np.int(i/2)][np.int(i%2)])

excluded = ['id','default_time','payoff_time','status_time']
categoricalColumns = ['REtype_CO_orig_time', 'REtype_PU_orig_time',
'REtype_SF_orig_time', 'investor_orig_time']
```

```
included = []
for c in mortgage.columns:
    if c not in excluded and c not in categoricalColumns:
        included.append(c)

# Here is a useful template to use for working with subplots.
#
######################################################################
fig, ax = plt.subplots(figsize=(20,40), ncols=2, nrows=8)

left   =  0.125  # the left side of the subplots of the figure
right  =  0.9    # the right side of the subplots of the figure
bottom =  0.1    # the bottom of the subplots of the figure
top    =  0.9    # the top of the subplots of the figure
wspace =  .5     # the amount of width reserved for blank space between subplots
hspace =  1.1    # the amount of height reserved for white space between subplots

# This function actually adjusts the sub plots using the above paramters
plt.subplots_adjust(
    left    =  left,
    bottom  =  bottom,
    right   =  right,
    top     =  top,
    wspace  =  wspace,
    hspace  =  hspace
)

# The amount of space above titles
y_title_margin = 1

plt.suptitle("Distribution of Features",
             y = 0.94, fontsize=15)
```

```
for i in range(0,len(included)):
    ax[np.int(i/2)][np.int(i%2)].set_ylabel("")
    ax[np.int(i/2)][np.int(i%2)].set_xlabel(included[i])
    k1=sns.distplot(mortgage[mortgage.default_time == 0][included[i]],
                color="darkgreen", ax=ax[np.int(i/2)][np.int(i%2)])

categoricalColumns = ['REtype_CO_orig_time', 'REtype_PU_orig_time',
'REtype_SF_orig_time', 'investor_orig_time']

######################################################################
fig, ax = plt.subplots(figsize=(20,15), ncols=2, nrows=2)

left   =  0.125  # the left side of the subplots of the figure
right  =  0.9    # the right side of the subplots of the figure
bottom =  0.1    # the bottom of the subplots of the figure
top    =  0.9    # the top of the subplots of the figure
wspace =  .5     # the amount of width reserved for blank space between subplots
hspace =  1.1    # the amount of height reserved for white space between subplots

# This function actually adjusts the sub plots using the above paramters
plt.subplots_adjust(
    left    =  left,
    bottom  =  bottom,
    right   =  right,
    top     =  top,
    wspace  =  wspace,
    hspace  =  hspace
)

# The amount of space above titles
y_title_margin = 1
```

```
plt.suptitle("Default Probability for Each Categorical Features",
             y = 0.94, fontsize=15)

for i in range(0,len(categoricalColumns)):
    g = sns.factorplot(x=categoricalColumns[i], y="default_time",
data=mortgage,ax=ax[np.int(i/2)][np.int(i%2)])
    # According to https://stackoverflow.com/questions/33925494/seaborn-produces-
separate-figures-in-subplots
    # factorplot creates a new FacetGrid instance (which in turn creates its own figure), on
which it will
    # apply a plotting function (pointplot by default). Because of this, If I did not close
factorplot with
    # following line, it also creates empty facetgrid
    plt.close(g.fig)

# I added this part so that heatmap starts with 'status_time'
corrList = ['default_time']

# id indicates which loan this specific record belongs, so I did not take it into account while
calculating correlation
# default_time and payoff_time are actually our target values splitted into two columns,
since status_time
# simply have the same information in single column, I chose it my target value and decide
not to use these two.
excluded = ['id','status_time','payoff_time']

for c in mortgage.columns:
    #status_time will be the first element in the heatmap
    if c != "default_time":
        if c not in excluded:
            corrList.append(c)
```

```
####################################
#corrMatrix = modelData[corrList].corr()
corrMatrix = mortgage[corrList].corr()

sns.set(font_scale=1.10)
plt.figure(figsize=(20, 20))

sns.heatmap(corrMatrix, vmax=.8, linewidths=0.01,
            square=True,annot=True,cmap='viridis',linecolor="white")
plt.title('Correlation between features');

print ("\033[1;35m"+"The Correlation with respect to the SalePrice\n--------------------------
------")
print(corrMatrix["default_time"].sort_values(ascending=False))

corrMatrix_status_time = corrMatrix["default_time"].sort_values(ascending=False)
width = 0
fig, ax = plt.subplots(figsize = (10,6))
rects = ax.barh(np.arange(len(corrMatrix_status_time)),
np.array(corrMatrix_status_time.values), color = 'red')
ax.set_yticks(np.arange(len(corrMatrix_status_time)) + ((width)/1))
ax.set_yticklabels(corrMatrix_status_time.index, rotation ='horizontal')
ax.set_xlabel("Correlation coefficient")
ax.set_title("Correlation Coefficients w.r.t Default",fontsize = 14);
ax.grid(True)

start = timeit.default_timer()

lgbm_params = {
    'random_state' : 1,
    'boosting_type':'gbdt',
    'objective':'binary',
    'metric':'binary_logloss',
    'n_estimators':675,
    }

lgbm_classifier = lightgbm.LGBMClassifier(**lgbm_params)
X = mortgage.drop(['default_time','payoff_time','status_time','id','time','orig_time'], axis =
1).copy()
y = mortgage['default_time'].copy()

X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2,random_state = 30)

lgbm_train_dataset = lightgbm.Dataset(X_train, label=y_train)
lgbm_classifier = lightgbm.train(params=lgbm_params,train_set=lgbm_train_dataset)

#Prediction
y_pred=lgbm_classifier.predict(X_test)

#convert into binary values
for i in range(0,y_pred.size):
    if y_pred[i]>=.5:       # setting threshold to .5
       y_pred[i]=1
    else:
       y_pred[i]=0

accuracy = accuracy_score(y_pred,y_test)
recall = recall_score(y_pred,y_test)
cnf_matrix = confusion_matrix(y_true=y_test, y_pred=y_pred)
stop = timeit.default_timer()

true_negative = cnf_matrix[0][0]
true_positive = cnf_matrix[1][1]
false_positive = cnf_matrix[0][1]
```

```
false_negative = cnf_matrix[1][0]

print('accuracy=',true_positive+true_negative/(true_positive+false_negative+true_negative
+false_positive)  )
print('recall=',true_positive/(true_positive+false_negative)  )

sns.heatmap(cnf_matrix,cmap="coolwarm_r",annot=True,linewidths=0.5)
plt.title("Confusion_matrix")
plt.xlabel("Predicted_Defaults")
plt.ylabel("Real_Defaults")
plt.show()



result_lgbm={'classifier':'LightGBM',
        'feature_reduction':'no',
        'numberofFeatures':len(X.columns),
        'Recall' : recall,
        'Accuracy' : accuracy,
        'elapsedTime' : stop-start }

print(result_lgbm)
del(X)
del(y)
del(X_train)
del(X_test)
del(y_train)
del(y_test)
del(lgbm_train_dataset)
```

```
underSampling_RD = mortgage.drop_duplicates(subset=['id'],keep='last')
sns.countplot("default_time",data=underSampling_RD)

# Put in our parameters for said classifiers
# Random Forest parameters
rf_params = {
    'random_state' : 1,
    'n_jobs': -1,
    'n_estimators': 500,
     'warm_start': True,
     #'max_features': 0.2,
    'max_depth': 6,
    'min_samples_leaf': 2,
    'max_features' : 'sqrt',
    'verbose': 0}

# Extra Trees Parameters
et_params = {
    'random_state' : 1,
    'n_jobs': -1,
    'n_estimators':500,
    #'max_features': 0.5,
    'max_depth': 8,
    'min_samples_leaf': 2,
    'verbose': 0}

# AdaBoost parameters
ada_params = {
    'random_state' : 1,
    'n_estimators': 500,
    'learning_rate' : 0.75}
```

```
# Gradient Boosting parameters
gb_params = {
    'random_state' : 1,
    'n_estimators': 500,
     #'max_features': 0.2,
    'max_depth': 5,
    'min_samples_leaf': 2,
    'verbose': 0}

# XG Boost parameters
xgb_params = {
    'random_state' : 1,
    'learning_rate':0.05,
    'n_estimators':500,
    'max_depth':3,
    'colsample_bytree':0.4}

# LightGBM
lgbm_params = {
    'random_state' : 1,
    'boosting_type':'gbdt',
    'objective':'binary',
    'metric':'binary_logloss',
    'n_estimators':675, }

# GaussianNB
gNB_params = {
    'random_state' : 1}
# DecisionTreeClassifier
dt_params = {
    'random_state' : 1, }
```

```
# KNeighbors Classifier
KNN_params = {
    'n_neighbors':10,
    'weights':'distance'}

# LinearDiscriminantAnalysis
lda_params = {
    'random_state' : 1}

# QuadraticDiscriminantAnalysis
qda_params = {
    'random_state' : 1}

# Logistic Regressor
lr_params = {
    'random_state' : 1 }

result_UnderSampling_RemoveDuplicate =[]
data = underSampling_RD.copy()
start = timeit.default_timer()

#Random Forest
X = data.drop(['default_time','payoff_time','status_time','id','time','orig_time'], axis =
1).copy()
y = data['default_time'].copy()
kfold= KFold(n_splits=10,random_state=42)
rf_classifier = RandomForestClassifier(**rf_params)
scoring= { 'acc':'accuracy',
            'rec':'recall',
            'prec':'precision',
            'f1':'f1', }
```

```
result = cross_validate(estimator=rf_classifier, X=X, y=y, cv = kfold, n_jobs=-
1,scoring=scoring,return_train_score=True)
stop = timeit.default_timer()

result_UnderSampling_RemoveDuplicate.append(
        {'classifier':'Random Forrest',
        'numberofFeatures':len(X.columns),
        'fit_time' : np.mean(result['fit_time']),
        'test_acc' : np.mean(result['test_acc']),
        'test_f1' : np.mean(result['test_f1']),
        'test_prec' : np.mean(result['test_prec']),
        'test_rec' : np.mean(result['test_rec']),
        'allScore' : result,
        'elapsedTime' : stop-start })

#Extra Trees
start = timeit.default_timer()

X = data.drop(['default_time','payoff_time','status_time','id','time','orig_time'], axis =
1).copy()
y = data['default_time'].copy()
kfold= KFold(n_splits=10,random_state=42)
et_classifier = ExtraTreesClassifier(**et_params)
scoring= {
        'acc':'accuracy',
        'rec':'recall',
        'prec':'precision',
        'f1':'f1',}
result = cross_validate(estimator=et_classifier, X=X, y=y, cv = kfold, n_jobs=-
1,scoring=scoring,return_train_score=True)
stop = timeit.default_timer()
```

```
result_UnderSampling_RemoveDuplicate.append(
        {'classifier':'Extra Trees',
        'numberofFeatures':len(X.columns),
        'fit_time' : np.mean(result['fit_time']),
        'test_acc' : np.mean(result['test_acc']),
        'test_f1' : np.mean(result['test_f1']),
        'test_prec' : np.mean(result['test_prec']),
        'test_rec' : np.mean(result['test_rec']),
        'allScore' : result,
        'elapsedTime' : stop-start })

#Adaboost
start = timeit.default_timer()

X = data.drop(['default_time','payoff_time','status_time','id','time','orig_time'], axis =
1).copy()
y = data['default_time'].copy()
kfold= KFold(n_splits=10,random_state=42)
ada_classifier = AdaBoostClassifier(**ada_params)
scoring= {
        'acc':'accuracy',
        'rec':'recall',
        'prec':'precision',
        'f1':'f1',}

result = cross_validate(estimator=ada_classifier, X=X, y=y, cv = kfold, n_jobs=-
1,scoring=scoring,return_train_score=True)
stop = timeit.default_timer()
result_UnderSampling_RemoveDuplicate.append(
        {'classifier':'AdaBoost',
        'numberofFeatures':len(X.columns),
        'fit_time' : np.mean(result['fit_time']),
        'test_acc' : np.mean(result['test_acc']),
        'test_f1' : np.mean(result['test_f1']),
        'test_prec' : np.mean(result['test_prec']),
        'test_rec' : np.mean(result['test_rec']),
        'allScore' : result,
        'elapsedTime' : stop-start })

#XGBoost
start = timeit.default_timer()
X = data.drop(['default_time','payoff_time','status_time','id','time','orig_time'], axis =
1).copy()
y = data['default_time'].copy()
kfold= KFold(n_splits=10,random_state=42)
xgb_classifier = XGBClassifier(**xgb_params)
scoring= {
        'acc':'accuracy',
        'rec':'recall',
        'prec':'precision',
        'f1':'f1', }
result = cross_validate(estimator=xgb_classifier, X=X, y=y, cv = kfold, n_jobs=-
1,scoring=scoring,return_train_score=True)
stop = timeit.default_timer()
result_UnderSampling_RemoveDuplicate.append(
        {'classifier':'XGBoost',
        'numberofFeatures':len(X.columns),
        'fit_time' : np.mean(result['fit_time']),
        'test_acc' : np.mean(result['test_acc']),
        'test_f1' : np.mean(result['test_f1']),
        'test_prec' : np.mean(result['test_prec']),
        'test_rec' : np.mean(result['test_rec'],
        'allScore' : result,
        'elapsedTime' : stop-start })
```

```
#GaussianNB
start = timeit.default_timer()
X = data.drop(['default_time','payoff_time','status_time','id','time','orig_time'], axis =
1).copy()
y = data['default_time'].copy()
kfold= KFold(n_splits=10,random_state=42)
#GaussianNB() does not take any parameter
gNB_classifier = GaussianNB()
scoring= {
        'acc':'accuracy',
        'rec':'recall',
        'prec':'precision',
        'f1':'f1',}
result = cross_validate(estimator=gNB_classifier, X=X, y=y, cv = kfold, n_jobs=-
1,scoring=scoring,return_train_score=True)
stop = timeit.default_timer()
result_UnderSampling_RemoveDuplicate.append(
        {'classifier':'GaussianNB',
        'numberofFeatures':len(X.columns),
        'fit_time' : np.mean(result['fit_time']),
        'test_acc' : np.mean(result['test_acc']),
        'test_f1' : np.mean(result['test_f1']),
        'test_prec' : np.mean(result['test_prec']),
        'test_rec' : np.mean(result['test_rec']),
        'allScore' : result,
        'elapsedTime' : stop-start })

#DecisionTreeClassifier
start = timeit.default_timer()
X = data.drop(['default_time','payoff_time','status_time','id','time','orig_time'], axis =
1).copy()
y = data['default_time'].copy()
```

```
kfold= KFold(n_splits=10,random_state=42)
dt_classifier = tree.DecisionTreeClassifier(**dt_params)
scoring= {
        'acc':'accuracy',
        'rec':'recall',
        'prec':'precision',
        'f1':'f1',  }

result = cross_validate(estimator=dt_classifier, X=X, y=y, cv = kfold, n_jobs=-
1,scoring=scoring,return_train_score=True)
stop = timeit.default_timer()

result_UnderSampling_RemoveDuplicate.append(
        {'classifier':'DecisionTree',
        'numberofFeatures':len(X.columns),
        'fit_time' : np.mean(result['fit_time']),
        'test_acc' : np.mean(result['test_acc']),
        'test_f1' : np.mean(result['test_f1']),
        'test_prec' : np.mean(result['test_prec']),
        'test_rec' : np.mean(result['test_rec']),
        'allScore' : result,
        'elapsedTime' : stop-start })

#KNeighbors Classifier
start = timeit.default_timer()
X = data.drop(['default_time','payoff_time','status_time','id','time','orig_time'], axis =
1).copy()
y = data['default_time'].copy()
kfold= KFold(n_splits=10,random_state=42)
KNN_classifier = KNeighborsClassifier(**KNN_params)
scoring= {
        'acc':'accuracy',
        'rec':'recall',
        'prec':'precision',
        'f1':'f1',  }
result = cross_validate(estimator=KNN_classifier, X=X, y=y, cv = kfold, n_jobs=-
1,scoring=scoring,return_train_score=True)
stop = timeit.default_timer()

result_UnderSampling_RemoveDuplicate.append(
        {'classifier':'KNeighbors',
        'numberofFeatures':len(X.columns),
        'fit_time' : np.mean(result['fit_time']),
        'test_acc' : np.mean(result['test_acc']),
        'test_f1' : np.mean(result['test_f1']),
        'test_prec' : np.mean(result['test_prec']),
        'test_rec' : np.mean(result['test_rec']),
        'allScore' : result,
        'elapsedTime' : stop-start })

#LinearDiscriminantAnalysis
start = timeit.default_timer()
X = data.drop(['default_time','payoff_time','status_time','id','time','orig_time'], axis =
1).copy()
y = data['default_time'].copy()

kfold= KFold(n_splits=10,random_state=42)
lda_classifier = LinearDiscriminantAnalysis()

scoring= {
        'acc':'accuracy',
        'rec':'recall',
        'prec':'precision',
        'f1':'f1', }
```

```
result = cross_validate(estimator=lda_classifier, X=X, y=y, cv = kfold, n_jobs=-
1,scoring=scoring,return_train_score=True)
stop = timeit.default_timer()

result_UnderSampling_RemoveDuplicate.append(
        {'classifier':'LinearDiscriminantAnalysis',
        'numberofFeatures':len(X.columns),
        'fit_time' : np.mean(result['fit_time']),
        'test_acc' : np.mean(result['test_acc']),
        'test_f1' : np.mean(result['test_f1']),
        'test_prec' : np.mean(result['test_prec']),
        'test_rec' : np.mean(result['test_rec']),
        'allScore' : result,
        'elapsedTime' : stop-start })

#QuadraticDiscriminantAnalysis
start = timeit.default_timer()
X = data.drop(['default_time','payoff_time','status_time','id','time','orig_time'], axis =
1).copy()
y = data['default_time'].copy()

kfold= KFold(n_splits=10,random_state=42)
qda_classifier = QuadraticDiscriminantAnalysis()

scoring= {
        'acc':'accuracy',
        'rec':'recall',
        'prec':'precision',
        'f1':'f1',}
result = cross_validate(estimator=qda_classifier, X=X, y=y, cv = kfold, n_jobs=-
1,scoring=scoring,return_train_score=True)
stop = timeit.default_timer()
```

```
result_UnderSampling_RemoveDuplicate.append(
        {'classifier':'QuadraticDiscriminantAnalysis',
        'numberofFeatures':len(X.columns),
        'fit_time' : np.mean(result['fit_time']),
        'test_acc' : np.mean(result['test_acc']),
        'test_f1' : np.mean(result['test_f1']),
        'test_prec' : np.mean(result['test_prec']),
        'test_rec' : np.mean(result['test_rec']),
        'allScore' : result,
        'elapsedTime' : stop-start })

#Logistic Regressor
start = timeit.default_timer()
X = data.drop(['default_time','payoff_time','status_time','id','time','orig_time'], axis =
1).copy()
y = data['default_time'].copy()
kfold= KFold(n_splits=10,random_state=42)
lr_classifier = LogisticRegression(**lr_params)

scoring= {
        'acc':'accuracy',
        'rec':'recall',
        'prec':'precision',
        'f1':'f1', }
result = cross_validate(estimator=lr_classifier, X=X, y=y, cv = kfold, n_jobs=-
1,scoring=scoring,return_train_score=True)
stop = timeit.default_timer()
result_UnderSampling_RemoveDuplicate.append(
        {'classifier':'Logistic Regression',
        'numberofFeatures':len(X.columns),
        'fit_time' : np.mean(result['fit_time']),
        'test_acc' : np.mean(result['test_acc']),
        'test_f1' : np.mean(result['test_f1']),
        'test_prec' : np.mean(result['test_prec']),
        'test_rec' : np.mean(result['test_rec']),
        'allScore' : result,
        'elapsedTime' : stop-start })

#Random Undersampling
data_default_indicies = mortgage[mortgage['default_time']==1].index
data_nondefault_indicies = mortgage[mortgage['default_time']!=1].index

#resolve class imbalance by randomly undersampling
total_number_of_default = len(data_default_indicies)
nondefault_undersampled_indicies = np.random.choice(data_nondefault_indicies,
total_number_of_default,replace=False)
undersampled_indicies =
np.concatenate([data_default_indicies,nondefault_undersampled_indicies])
undersampled_indicies
data = mortgage.loc[undersampled_indicies,:]
result_UnderSampling_Random =[]
sns.countplot("default_time",data=data)

#Random Forest
start = timeit.default_timer()
X = data.drop(['default_time','payoff_time','status_time','id','time','orig_time'], axis =
1).copy()
y = data['default_time'].copy()

kfold= KFold(n_splits=10,random_state=42)
rf_classifier = RandomForestClassifier(**rf_params)
```

```
scoring= {
        'acc':'accuracy',
        'rec':'recall',
        'prec':'precision',
        'f1':'f1',}

result = cross_validate(estimator=rf_classifier, X=X, y=y, cv = kfold, n_jobs=-
1,scoring=scoring,return_train_score=True)
stop = timeit.default_timer()

result_UnderSampling_Random.append(
        {'classifier':'Random Forrest',
        'numberofFeatures':len(X.columns),
        'fit_time' : np.mean(result['fit_time']),
        'test_acc' : np.mean(result['test_acc']),
        'test_f1' : np.mean(result['test_f1']),
        'test_prec' : np.mean(result['test_prec']),
        'test_rec' : np.mean(result['test_rec']),
        'allScore' : result,
        'elapsedTime' : stop-start })

#Extra Trees
start = timeit.default_timer()

X = data.drop(['default_time','payoff_time','status_time','id','time','orig_time'], axis =
1).copy()
y = data['default_time'].copy()
kfold= KFold(n_splits=10,random_state=42)
et_classifier = ExtraTreesClassifier(**et_params)
```

```
scoring= {
        'acc':'accuracy',
        'rec':'recall',
        'prec':'precision',
        'f1':'f1', }

result = cross_validate(estimator=et_classifier, X=X, y=y, cv = kfold, n_jobs=-
1,scoring=scoring,return_train_score=True)
stop = timeit.default_timer()

result_UnderSampling_Random.append(
        {'classifier':'Extra Trees',
        'numberofFeatures':len(X.columns),
        'fit_time' : np.mean(result['fit_time']),
        'test_acc' : np.mean(result['test_acc']),
        'test_f1' : np.mean(result['test_f1']),
        'test_prec' : np.mean(result['test_prec']),
        'test_rec' : np.mean(result['test_rec']),
        'allScore' : result,
        'elapsedTime' : stop-start })

#AdaBoost
start = timeit.default_timer()

X = data.drop(['default_time','payoff_time','status_time','id','time','orig_time'], axis =
1).copy()
y = data['default_time'].copy()
kfold= KFold(n_splits=10,random_state=42)
ada_classifier = AdaBoostClassifier(**ada_params)
```

```
scoring= {
        'acc':'accuracy',
        'rec':'recall',
        'prec':'precision',
        'f1':'f1', }

result = cross_validate(estimator=ada_classifier, X=X, y=y, cv = kfold, n_jobs=-
1,scoring=scoring,return_train_score=True)
stop = timeit.default_timer()

result_UnderSampling_Random.append(
        {'classifier':'AdaBoost',
        'numberofFeatures':len(X.columns),
        'fit_time' : np.mean(result['fit_time']),
        'test_acc' : np.mean(result['test_acc']),
        'test_f1' : np.mean(result['test_f1']),
        'test_prec' : np.mean(result['test_prec']),
        'test_rec' : np.mean(result['test_rec']),
        'allScore' : result,
        'elapsedTime' : stop-start })

#XGBoost
start = timeit.default_timer()
X = data.drop(['default_time','payoff_time','status_time','id','time','orig_time'], axis =
1).copy()
y = data['default_time'].copy()
kfold= KFold(n_splits=10,random_state=42)
xgb_classifier = XGBClassifier(**xgb_params)
```

```
scoring= {
        'acc':'accuracy',
        'rec':'recall',
        'prec':'precision',
        'f1':'f1',}
result = cross_validate(estimator=xgb_classifier, X=X, y=y, cv = kfold, n_jobs=-
1,scoring=scoring,return_train_score=True)
stop = timeit.default_timer()

result_UnderSampling_Random.append(
        {'classifier':'XGBoost',
        'numberofFeatures':len(X.columns),
        'fit_time' : np.mean(result['fit_time']),
        'test_acc' : np.mean(result['test_acc']),
        'test_f1' : np.mean(result['test_f1']),
        'test_prec' : np.mean(result['test_prec']),
        'test_rec' : np.mean(result['test_rec']),
        'allScore' : result,
        'elapsedTime' : stop-start })

#GaussianNB
start = timeit.default_timer()
X = data.drop(['default_time','payoff_time','status_time','id','time','orig_time'], axis =
1).copy()
y = data['default_time'].copy()
kfold= KFold(n_splits=10,random_state=42)
#GaussianNB() does not take any parameter
gNB_classifier = GaussianNB()
```

```
scoring= {
        'acc':'accuracy',
        'rec':'recall',
        'prec':'precision',
        'f1':'f1',}
result = cross_validate(estimator=gNB_classifier, X=X, y=y, cv = kfold, n_jobs=-
1,scoring=scoring,return_train_score=True)
stop = timeit.default_timer()

result_UnderSampling_Random.append(
        {'classifier':'GaussianNB',
        'numberofFeatures':len(X.columns),
        'fit_time' : np.mean(result['fit_time']),
        'test_acc' : np.mean(result['test_acc']),
        'test_f1' : np.mean(result['test_f1']),
        'test_prec' : np.mean(result['test_prec']),
        'test_rec' : np.mean(result['test_rec']),
        'allScore' : result,
        'elapsedTime' : stop-start })

#DecisionTreeClassifier
start = timeit.default_timer()
X = data.drop(['default_time','payoff_time','status_time','id','time','orig_time'], axis =
1).copy()
y = data['default_time'].copy()
kfold= KFold(n_splits=10,random_state=42)
dt_classifier = tree.DecisionTreeClassifier(**dt_params)

scoring= {
        'acc':'accuracy',
        'rec':'recall',
        'prec':'precision',
        'f1':'f1',}
result = cross_validate(estimator=dt_classifier, X=X, y=y, cv = kfold, n_jobs=-
1,scoring=scoring,return_train_score=True)
stop = timeit.default_timer()

result_UnderSampling_Random.append(
        {'classifier':'DecisionTree',
        'numberofFeatures':len(X.columns),
        'fit_time' : np.mean(result['fit_time']),
        'test_acc' : np.mean(result['test_acc']),
        'test_f1' : np.mean(result['test_f1']),
        'test_prec' : np.mean(result['test_prec']),
        'test_rec' : np.mean(result['test_rec']),
        'allScore' : result,
        'elapsedTime' : stop-start })

#KNeighbors Classifier
start = timeit.default_timer()
X = data.drop(['default_time','payoff_time','status_time','id','time','orig_time'], axis =
1).copy()
y = data['default_time'].copy()
kfold= KFold(n_splits=10,random_state=42)
KNN_classifier = KNeighborsClassifier(**KNN_params)
scoring= {
        'acc':'accuracy',
        'rec':'recall',
        'prec':'precision',
        'f1':'f1',}
result = cross_validate(estimator=KNN_classifier, X=X, y=y, cv = kfold, n_jobs=-
1,scoring=scoring,return_train_score=True)
stop = timeit.default_timer()
```

```
result_UnderSampling_Random.append(
        {'classifier':'KNeighbors',
        'numberofFeatures':len(X.columns),
        'fit_time' : np.mean(result['fit_time']),
        'test_acc' : np.mean(result['test_acc']),
        'test_f1' : np.mean(result['test_f1']),
        'test_prec' : np.mean(result['test_prec']),
        'test_rec' : np.mean(result['test_rec']),
        'allScore' : result,
        'elapsedTime' : stop-start })

#LinearDiscriminantAnalysis
start = timeit.default_timer()

X = data.drop(['default_time','payoff_time','status_time','id','time','orig_time'], axis =
1).copy()
y = data['default_time'].copy()
kfold= KFold(n_splits=10,random_state=42)
lda_classifier = LinearDiscriminantAnalysis()
scoring= {
        'acc':'accuracy',
        'rec':'recall',
        'prec':'precision',
        'f1':'f1', }
result = cross_validate(estimator=lda_classifier, X=X, y=y, cv = kfold, n_jobs=-
1,scoring=scoring,return_train_score=True)
stop = timeit.default_timer()

result_UnderSampling_Random.append(
        {'classifier':'LinearDiscriminantAnalysis',
        'numberofFeatures':len(X.columns),
        'fit_time' : np.mean(result['fit_time']),
            'test_acc' : np.mean(result['test_acc']),
            'test_f1' : np.mean(result['test_f1']),
            'test_prec' : np.mean(result['test_prec']),
            'test_rec' : np.mean(result['test_rec']),
            'allScore' : result,
            'elapsedTime' : stop-start })

#QuadraticDiscriminantAnalysis
start = timeit.default_timer()

X = data.drop(['default_time','payoff_time','status_time','id','time','orig_time'], axis =
1).copy()
y = data['default_time'].copy()
kfold= KFold(n_splits=10,random_state=42)
qda_classifier = QuadraticDiscriminantAnalysis()

scoring= {
            'acc':'accuracy',
            'rec':'recall',
            'prec':'precision',
            'f1':'f1', }

result = cross_validate(estimator=qda_classifier, X=X, y=y, cv = kfold, n_jobs=-
1,scoring=scoring,return_train_score=True)
stop = timeit.default_timer()

result_UnderSampling_Random.append(
            {'classifier':'QuadraticDiscriminantAnalysis',
            'numberofFeatures':len(X.columns),
            'fit_time' : np.mean(result['fit_time']),
            'test_acc' : np.mean(result['test_acc']),
            'test_f1' : np.mean(result['test_f1']),
            'test_prec' : np.mean(result['test_prec']),
            'test_rec' : np.mean(result['test_rec']),
            'allScore' : result,
            'elapsedTime' : stop-start })

#Logistic Regressor
start = timeit.default_timer()

X = data.drop(['default_time','payoff_time','status_time','id','time','orig_time'], axis =
1).copy()
y = data['default_time'].copy()
kfold= KFold(n_splits=10,random_state=42)
lr_classifier = LogisticRegression(**lr_params)

scoring= {
            'acc':'accuracy',
            'rec':'recall',
            'prec':'precision',
            'f1':'f1',}

result = cross_validate(estimator=lr_classifier, X=X, y=y, cv = kfold, n_jobs=-
1,scoring=scoring,return_train_score=True)
stop = timeit.default_timer()

result_UnderSampling_Random.append(
            {'classifier':'Logistic Regression',
            'numberofFeatures':len(X.columns),
            'fit_time' : np.mean(result['fit_time']),
            'test_acc' : np.mean(result['test_acc']),
            'test_f1' : np.mean(result['test_f1']),
            'test_prec' : np.mean(result['test_prec']),
            'test_rec' : np.mean(result['test_rec']),
          'allScore' : result,
          'elapsedTime' : stop-start })
# Resampling: OverSampling
# SMOTE
data = mortgage.copy()
result_Weighted =[]

X = data.drop(['default_time','payoff_time','status_time','id','time','orig_time'], axis =
1).copy()
y = data['default_time'].copy()
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2,random_state = 30)
sm = SMOTE(random_state=12,ratio='minority')
X_train_res, y_train_res = sm.fit_sample(X_train,y_train)
result_OverSampling_SMOTE = []

temp = X_train_res.copy()
temp = pd.DataFrame(temp)
temp['default_time'] = y_train_res
sns.countplot("default_time",data=temp)

# Random Forest
start = timeit.default_timer()

rf_classifier = RandomForestClassifier(**rf_params)
rf_classifier.fit(X_train_res, y_train_res)
#Prediction
y_pred=rf_classifier.predict(X_test)
cnf_matrix = confusion_matrix(y_true=y_test, y_pred=y_pred)
stop = timeit.default_timer()
true_negative = cnf_matrix[0][0]
true_positive = cnf_matrix[1][1]
```

```
false_positive = cnf_matrix[0][1]
false_negative = cnf_matrix[1][0]

sns.heatmap(cnf_matrix,cmap="coolwarm_r",annot=True,linewidths=0.5)
plt.title("Confusion_matrix")
plt.xlabel("Predicted_Defaults")
plt.ylabel("Real_Defaults")
plt.show()

result_OverSampling_SMOTE.append(
        {'classifier':'Random Forrest',
        'numberofFeatures':len(X.columns),
        'fit_time' : stop-start,
        'test_acc' : accuracy_score(y_test,y_pred),
        'test_f1' : f1_score(y_test,y_pred),
        'test_prec' : precision_score(y_test,y_pred),
        'test_rec' : recall_score(y_test,y_pred),
        'test_roc' : roc_auc_score(y_test,y_pred),
        'allScore' : None,
        'elapsedTime' : None })
result_OverSampling_SMOTE

# Extra Trees
start = timeit.default_timer()
et_classifier = ExtraTreesClassifier(**et_params)
et_classifier.fit(X_train_res, y_train_res)
#Prediction
y_pred=et_classifier.predict(X_test)
cnf_matrix = confusion_matrix(y_true=y_test, y_pred=y_pred)
stop = timeit.default_timer()
true_negative = cnf_matrix[0][0]
true_positive = cnf_matrix[1][1]
```

```
false_positive = cnf_matrix[0][1]
false_negative = cnf_matrix[1][0]

sns.heatmap(cnf_matrix,cmap="coolwarm_r",annot=True,linewidths=0.5)
plt.title("Confusion_matrix")
plt.xlabel("Predicted_Defaults")
plt.ylabel("Real_Defaults")
plt.show()

result_OverSampling_SMOTE.append(
        {'classifier':'Extra Trees',
        'numberofFeatures':len(X.columns),
        'fit_time' : stop-start,
        'test_acc' : accuracy_score(y_test,y_pred),
        'test_f1' : f1_score(y_test,y_pred),
        'test_prec' : precision_score(y_test,y_pred),
        'test_rec' : recall_score(y_test,y_pred),
        'test_roc' : roc_auc_score(y_test,y_pred),
        'allScore' : None,
        'elapsedTime' : None })

#AdaBoost
start = timeit.default_timer()
ada_classifier = AdaBoostClassifier(**ada_params)
ada_classifier.fit(X_train_res, y_train_res)
#Prediction
y_pred=ada_classifier.predict(X_test)
cnf_matrix = confusion_matrix(y_true=y_test, y_pred=y_pred)
stop = timeit.default_timer()
true_negative = cnf_matrix[0][0]
true_positive = cnf_matrix[1][1]
false_positive = cnf_matrix[0][1]
```

```
false_negative = cnf_matrix[1][0]

sns.heatmap(cnf_matrix,cmap="coolwarm_r",annot=True,linewidths=0.5)
plt.title("Confusion_matrix")
plt.xlabel("Predicted_Defaults")
plt.ylabel("Real_Defaults")
plt.show()

result_OverSampling_SMOTE.append(
        {'classifier':'AdaBoost',
        'numberofFeatures':len(X.columns),
        'fit_time' : stop-start,
        'test_acc' : accuracy_score(y_test,y_pred),
        'test_f1' : f1_score(y_test,y_pred),
        'test_prec' : precision_score(y_test,y_pred),
        'test_rec' : recall_score(y_test,y_pred),
        'test_roc' : roc_auc_score(y_test,y_pred),
        'allScore' : None,
        'elapsedTime' : None })

#XGBoost
start = timeit.default_timer()
xgb_classifier = XGBClassifier(**xgb_params)
xgb_classifier.fit(X_train_res, y_train_res)
#Prediction
y_pred=xgb_classifier.predict(X_test.values)
cnf_matrix = confusion_matrix(y_true=y_test, y_pred=y_pred)
stop = timeit.default_timer()
true_negative = cnf_matrix[0][0]
true_positive = cnf_matrix[1][1]
false_positive = cnf_matrix[0][1]
false_negative = cnf_matrix[1][0]
```

```
78sns.heatmap(cnf_matrix,cmap="coolwarm_r",annot=True,linewidths=0.5)
plt.title("Confusion_matrix")
plt.xlabel("Predicted_Defaults")
plt.ylabel("Real_Defaults")
plt.show()

result_OverSampling_SMOTE.append(
        {'classifier':'XGBoost',
        'numberofFeatures':len(X.columns),
        'fit_time' : stop-start,
        'test_acc' : accuracy_score(y_test,y_pred),
        'test_f1' : f1_score(y_test,y_pred),
        'test_prec' : precision_score(y_test,y_pred),
        'test_rec' : recall_score(y_test,y_pred),
        'test_roc' : roc_auc_score(y_test,y_pred),
        'allScore' : None,
        'elapsedTime' : None })

#GaussianNB
start = timeit.default_timer()
gNB_classifier = GaussianNB()
gNB_classifier.fit(X_train_res, y_train_res)
#Prediction
y_pred=gNB_classifier.predict(X_test)
cnf_matrix = confusion_matrix(y_true=y_test, y_pred=y_pred)
stop = timeit.default_timer()

true_negative = cnf_matrix[0][0]
true_positive = cnf_matrix[1][1]
false_positive = cnf_matrix[0][1]
false_negative = cnf_matrix[1][0]
```

```
sns.heatmap(cnf_matrix,cmap="coolwarm_r",annot=True,linewidths=0.5)
plt.title("Confusion_matrix")
plt.xlabel("Predicted_Defaults")
plt.ylabel("Real_Defaults")
plt.show()

result_OverSampling_SMOTE.append(
        {'classifier':'GaussianNB',
        'numberofFeatures':len(X.columns),
        'fit_time' : stop-start,
        'test_acc' : accuracy_score(y_test,y_pred),
        'test_f1' : f1_score(y_test,y_pred),
        'test_prec' : precision_score(y_test,y_pred),
        'test_rec' : recall_score(y_test,y_pred),
        'test_roc' : roc_auc_score(y_test,y_pred),
        'allScore' : None,
        'elapsedTime' : None })

DecisionTreeClassifier
start = timeit.default_timer()
dt_classifier = tree.DecisionTreeClassifier(**dt_params)
dt_classifier.fit(X_train_res, y_train_res)
#Prediction
y_pred=dt_classifier.predict(X_test)
cnf_matrix = confusion_matrix(y_true=y_test, y_pred=y_pred)
stop = timeit.default_timer()

true_negative = cnf_matrix[0][0]
true_positive = cnf_matrix[1][1]
false_positive = cnf_matrix[0][1]
false_negative = cnf_matrix[1][0]
```

```
sns.heatmap(cnf_matrix,cmap="coolwarm_r",annot=True,linewidths=0.5)
plt.title("Confusion_matrix")
plt.xlabel("Predicted_Defaults")
plt.ylabel("Real_Defaults")
plt.show()

result_OverSampling_SMOTE.append(
        {'classifier':'DecisionTree',
        'numberofFeatures':len(X.columns),
        'fit_time' : stop-start,
        'test_acc' : accuracy_score(y_test,y_pred),
        'test_f1' : f1_score(y_test,y_pred),
        'test_prec' : precision_score(y_test,y_pred),
        'test_rec' : recall_score(y_test,y_pred),
        'test_roc' : roc_auc_score(y_test,y_pred),
        'allScore' : None,
        'elapsedTime' : None })

#KNeighbors Classifier
start = timeit.default_timer()

KNN_classifier = KNeighborsClassifier(**KNN_params)
KNN_classifier.fit(X_train_res, y_train_res)
#Prediction
y_pred=KNN_classifier.predict(X_test)
cnf_matrix = confusion_matrix(y_true=y_test, y_pred=y_pred)
stop = timeit.default_timer()
true_negative = cnf_matrix[0][0]
true_positive = cnf_matrix[1][1]
false_positive = cnf_matrix[0][1]
false_negative = cnf_matrix[1][0]
```

```
sns.heatmap(cnf_matrix,cmap="coolwarm_r",annot=True,linewidths=0.5)
plt.title("Confusion_matrix")
plt.xlabel("Predicted_Defaults")
plt.ylabel("Real_Defaults")
plt.show()

result_OverSampling_SMOTE.append(
        {'classifier':'KNeighbors',
        'numberofFeatures':len(X.columns),
        'fit_time' : stop-start,
        'test_acc' : accuracy_score(y_test,y_pred),
        'test_f1' : f1_score(y_test,y_pred),
        'test_prec' : precision_score(y_test,y_pred),
        'test_rec' : recall_score(y_test,y_pred),
        'test_roc' : roc_auc_score(y_test,y_pred),
        'allScore' : None,
        'elapsedTime' : None })

# LinearDiscriminantAnalysis
start = timeit.default_timer()
lda_classifier = LinearDiscriminantAnalysis()
lda_classifier.fit(X_train_res, y_train_res)
#Prediction
y_pred=lda_classifier.predict(X_test)
cnf_matrix = confusion_matrix(y_true=y_test, y_pred=y_pred)
stop = timeit.default_timer()

true_negative = cnf_matrix[0][0]
true_positive = cnf_matrix[1][1]
false_positive = cnf_matrix[0][1]
false_negative = cnf_matrix[1][0]
```

```
sns.heatmap(cnf_matrix,cmap="coolwarm_r",annot=True,linewidths=0.5)
plt.title("Confusion_matrix")
plt.xlabel("Predicted_Defaults")
plt.ylabel("Real_Defaults")
plt.show()

result_OverSampling_SMOTE.append(
        {'classifier':'LinearDiscriminantAnalysis',
        'numberofFeatures':len(X.columns),
        'fit_time' : stop-start,
        'test_acc' : accuracy_score(y_test,y_pred),
        'test_f1' : f1_score(y_test,y_pred),
        'test_prec' : precision_score(y_test,y_pred),
        'test_rec' : recall_score(y_test,y_pred),
        'test_roc' : roc_auc_score(y_test,y_pred),
        'allScore' : None,
        'elapsedTime' : None })

# QuadraticDiscriminantAnalysis
start = timeit.default_timer()
qda_classifier = QuadraticDiscriminantAnalysis()
qda_classifier.fit(X_train_res, y_train_res)
#Prediction
y_pred=qda_classifier.predict(X_test)
cnf_matrix = confusion_matrix(y_true=y_test, y_pred=y_pred)
stop = timeit.default_timer()
true_negative = cnf_matrix[0][0]
true_positive = cnf_matrix[1][1]
false_positive = cnf_matrix[0][1]
false_negative = cnf_matrix[1][0]
```

```
sns.heatmap(cnf_matrix,cmap="coolwarm_r",annot=True,linewidths=0.5)
plt.title("Confusion_matrix")
plt.xlabel("Predicted_Defaults")
plt.ylabel("Real_Defaults")
plt.show()

result_OverSampling_SMOTE.append(
        {'classifier':'QuadraticDiscriminantAnalysis',
        'numberofFeatures':len(X.columns),
        'fit_time' : stop-start,
        'test_acc' : accuracy_score(y_test,y_pred),
        'test_f1' : f1_score(y_test,y_pred),
        'test_prec' : precision_score(y_test,y_pred),
        'test_rec' : recall_score(y_test,y_pred),
        'test_roc' : roc_auc_score(y_test,y_pred),
        'allScore' : None,
        'elapsedTime' : None })

#Logistic Regressor
start = timeit.default_timer()

lr_classifier = LogisticRegression(**lr_params)
lr_classifier.fit(X_train_res, y_train_res)
#Prediction
y_pred=lr_classifier.predict(X_test)
cnf_matrix = confusion_matrix(y_true=y_test, y_pred=y_pred)
stop = timeit.default_timer()

true_negative = cnf_matrix[0][0]
true_positive = cnf_matrix[1][1]
false_positive = cnf_matrix[0][1]
false_negative = cnf_matrix[1][0]
```

```
sns.heatmap(cnf_matrix,cmap="coolwarm_r",annot=True,linewidths=0.5)
plt.title("Confusion_matrix")
plt.xlabel("Predicted_Defaults")
plt.ylabel("Real_Defaults")
plt.show()

result_OverSampling_SMOTE.append(
        {'classifier':'Logistic Regression',
        'numberofFeatures':len(X.columns),
        'fit_time' : stop-start,
        'test_acc' : accuracy_score(y_test,y_pred),
        'test_f1' : f1_score(y_test,y_pred),
        'test_prec' : precision_score(y_test,y_pred),
        'test_rec' : recall_score(y_test,y_pred),
        'test_roc' : roc_auc_score(y_test,y_pred),
        'allScore' : None,
        'elapsedTime' : None })

# Results
result_UnderSampling_RemoveDuplicate_DF =
pd.DataFrame(result_UnderSampling_RemoveDuplicate)
result_UnderSampling_RemoveDuplicate_DF

result_UnderSampling_Random_DF = pd.DataFrame(result_UnderSampling_Random)
result_UnderSampling_Random_DF

result_OverSampling_SMOTE_DF = pd.DataFrame(result_OverSampling_SMOTE)
result_OverSampling_SMOTE_DF
```

```
# saved results for later use.
result_UnderSampling_RemoveDuplicate_DF.to_pickle('result_UnderSampling_Remove
Duplicate_DF')
result_UnderSampling_Random_DF.to_pickle('result_UnderSampling_Random_DF.pckl')
result_OverSampling_SMOTE_DF.to_pickle('result_OverSampling_SMOTE_DF.pckl')
result_Weighted_DF.to_pickle('result_Weighted_DF.pckl')

# Model Comparison: UnderSampling | Duplicates were removed
fig, ax = plt.subplots(figsize=(15,8))
ind = np.arange(result_UnderSampling_RemoveDuplicate_DF.shape[0])
width = 0.1
rects1 = ax.bar(ind - 3*width/2,result_UnderSampling_RemoveDuplicate_DF['test_acc'] ,
width, color='teal', label='Accuracy')
rects2 = ax.bar(ind - width/2,result_UnderSampling_RemoveDuplicate_DF['test_rec'] ,
width, color='crimson', label='Recall')
rects3 = ax.bar(ind + width/2,result_UnderSampling_RemoveDuplicate_DF['test_prec'] ,
width, color='orange', label='Precision')
rects4 = ax.bar(ind + 3*width/2,result_UnderSampling_RemoveDuplicate_DF['test_f1'] ,
width, color='salmon', label='f1')
tickLabels = result_UnderSampling_RemoveDuplicate_DF['classifier'].tolist()

ax.grid(color='g', linestyle='-', linewidth=0.1)
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
ax.set_xticklabels(tickLabels)
ax.set_title('UnderSampling | Duplicates were removed')
ax.set_facecolor('white')
plt.xticks(np.arange(0, 10, 1.0),rotation=45)
plt.show()
```

```
# Model Comparison: UnderSampling | Random subset
fig, ax = plt.subplots(figsize=(15,8))
ind = np.arange(result_UnderSampling_Random_DF.shape[0])
width = 0.1
rects1 = ax.bar(ind - 3*width/2,result_UnderSampling_Random_DF['test_acc'] , width,
color='teal', label='Accuracy')
rects2 = ax.bar(ind - width/2,result_UnderSampling_Random_DF['test_rec'] , width,
color='crimson', label='Recall')
rects3 = ax.bar(ind + width/2,result_UnderSampling_Random_DF['test_prec'] , width,
color='orange', label='Precision')
rects4 = ax.bar(ind + 3*width/2,result_UnderSampling_Random_DF['test_f1'] , width,
color='salmon', label='f1')
tickLabels = result_UnderSampling_Random_DF['classifier'].tolist()

ax.grid(color='g', linestyle='-', linewidth=0.1)
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
ax.set_xticklabels(tickLabels)
ax.set_title('UnderSampling | Random subset of nondefault records with the size of default
ones')
ax.set_facecolor('white')
plt.xticks(np.arange(0, 10, 1.0),rotation=45)
plt.show()

# Model Comparison: OverSampling | SMOTE
fig, ax = plt.subplots(figsize=(15,8))
ind = np.arange(result_OverSampling_SMOTE_DF.shape[0])
width = 0.1
rects1 = ax.bar(ind - 3*width/2,result_OverSampling_SMOTE_DF['test_acc'] , width,
color='teal', label='Accuracy')
rects2 = ax.bar(ind - width/2,result_OverSampling_SMOTE_DF['test_rec'] , width,
color='crimson', label='Recall')
```

```
rects3 = ax.bar(ind + width/2,result_OverSampling_SMOTE_DF['test_prec'] , width,
color='orange', label='Precision')
rects4 = ax.bar(ind + 3*width/2,result_OverSampling_SMOTE_DF['test_f1'] , width,
color='salmon', label='f1')
tickLabels = result_OverSampling_SMOTE_DF['classifier'].tolist()

ax.grid(color='g', linestyle='-', linewidth=0.1)
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
ax.set_xticklabels(tickLabels)
ax.set_title('OverSampling | SMOTE')
ax.set_facecolor('white')
plt.xticks(np.arange(0, 10, 1.0),rotation=45)
plt.show()

# Speed Comparison: UnderSampling | Duplicates were removed
fig, ax = plt.subplots(figsize=(15,8))
width = 0.1
rects1 = ax.bar(np.arange(0, 10, 1.0),
            result_UnderSampling_RemoveDuplicate_DF['fit_time'] ,
            width, color='teal', label='UnderSampling | Duplicates were removed')

tickLabels = result_UnderSampling_RemoveDuplicate_DF['classifier'].tolist(
ax.grid(color='g', linestyle='-', linewidth=0.1)
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
ax.set_xticklabels(tickLabels)
ax.set_title('Fit Time')
ax.set_facecolor('white')
plt.xticks(np.arange(0, 10, 1.0),rotation=45)
plt.show()
```

```
# Speed Comparison: UnderSampling | Random subset
fig, ax = plt.subplots(figsize=(15,8))
width = 0.1
rects2 = ax.bar(np.arange(0, 10, 1.0),
            result_UnderSampling_Random_DF['fit_time'] ,
            width, color='crimson', label='UnderSampling | Random subset')
tickLabels = result_UnderSampling_Random_DF['classifier'].tolist()

ax.grid(color='g', linestyle='-', linewidth=0.1)
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
ax.set_xticklabels(tickLabels)
ax.set_title('Fit Time')
ax.set_facecolor('white')
plt.xticks(np.arange(0, 10, 1.0),rotation=45)
plt.show()

# Model Comparison: OverSampling | SMOTE
fig, ax = plt.subplots(figsize=(15,8))
width = 0.1
rects3 = ax.bar(np.arange(0, 10, 1.0),
            result_OverSampling_SMOTE_DF['fit_time'] ,
            width, color='orange', label='OverSampling | SMOTE')

tickLabels = result_OverSampling_SMOTE_DF['classifier'].tolist()

ax.grid(color='g', linestyle='-', linewidth=0.1)
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
ax.set_xticklabels(tickLabels)
ax.set_title('Fit Time')
ax.set_facecolor('white')
plt.xticks(np.arange(0, 10, 1.0),rotation=45)
plt.show()
```

APPENDIX B

Github link: https://github.com/itezgiden/capstone/blob/master/Capstone.ipynb

MEF UNIVERSITY

Name of the project: Default Prediction Models For Mortgage Loans
Name/Last Name of the Student: Ilknur Tezgiden
Date of Thesis Defense: 03/09/2018

I hereby state that the graduation project prepared by Ilknur Tezgiden has been completed under my supervision. I accept this work as a "Graduation Project".

03/09/2018
Dr. Berk Orbay

I hereby state that I have examined this graduation project by Ilknur Tezgiden which is accepted by his supervisor. This work is acceptable as a graduation project and the student is eligible to take the graduation project examination.

03/09/2018

Prof. Özgür Özlük
[illegible]
Director
of
Big Data Analytics Program

We hereby state that we have held the graduation examination of and agree that the student has satisfied all requirements.

THE EXAMINATION COMMITTEE

Committee Member	Signature
1. Dr. Berk Orbay	[illegible].................................
2. Prof. Özgür Özlük	[illegible].................................

Academic Honesty Pledge

I promise not to collaborate with anyone, not to seek or accept any outside help, and not to give any help to others.

I understand that all resources in print or on the web must be explicitly cited.

In keeping with MEF University's ideals, I pledge that this work is my own and that I have neither given nor received inappropriate assistance in preparing it.

Name	Date	Signature
Ilknur Tezgiden	03/09/2018	[illegible]