Project Report

Home Loan Repayment Prediction

Devangi Rajput
Computer Engineering
Stevens Institute of Technology
NJ, USA
drajput4@stevens.edu

ABSTRACT

Financial institutions have to consider several variables while approving a loan. To determine whether a given borrower can fully pay off the loan or cause it to be charged off is difficult. If the lender is too strict, fewer loans get approved, which means there is less interest to collect. But if they are too lax, they end up approving loans that default. Machine learning can help us predict which loans will be charged off. Through various machine learning methods this project gives the predictive patterns in the financial data that can be used to ensure the clients that are capable of repayment are not rejected.

Table of Contents:

1. INTRODUCTION:

A. Overview:

The two most critical questions in the lending industry are: 1) How risky is the borrower? 2) Given the borrower's risk, should we lend him/her? The answer to the first question determines the interest rate the borrower would have. Interest rate measures among other things (such as time value of money) the riskiness of the borrower, i.e. the riskier the borrower, the higher the interest rate. With interest rate in mind, we can then determine if the borrower is eligible for the loan.

Investors (lenders) provide loans to borrowers in exchange for the promise of repayment with interest. That means the lender only makes profit (interest) if the borrower pays off the loan. However, if he/she doesn't repay the loan, then the lender loses money.

B. Objective:

The primary objective of this project is to provide accurate predictions of whether a customer will be able to repay his/her home loan. For this project, Linear Models like Logistic Regression and Naïve Bayes, Non-Linear Models like K-Nearest Neighbors and Tree based models like Decision Trees, Random Forest and Gradient Boosting are used to predict the repayment potentiality.

C. Project Flow:

- **Understanding the dataset:**

We have a training and testing dataset containing features. The training dataset has the target variable labeled as '1' and '0'. We need to predict this target variable for the test dataset.

- **Data Pre-processing:**

This step involved importing the csv files and constructing a data-frame. Pandas were used to create a data-frame. The data is then analyzed to remove all the missing values. Further, the unique values of each feature for train and test dataset is identified. In brief, data pre-processing steps will include loading and reading the dataset and making it ready for training.

- **Implementation of Machine Learning in brief:**

I. **Training**: For training, techniques like One-Hot Encoding are deployed. Machine learning algorithms cannot work with categorical data directly. Categorical data are variables that contain label values rather than numeric values. Categorical variables are often called *nominal*. Nominal must be converted to numbers. Therefore, one-hot encoding is performed on both training and test datasets.

II. **Testing**: After training the models, we evaluate the predictions using the test dataset. This will let us know how the models will be training and decide on whether we should modify its structure or hyperparameters. Then we use these final trained models to predict whether a customer is most likely to repay his loan.

2. IMPLEMENTATION:

This project is done in various stages illustrated as below. Python Libraries like Scikit-Learn and Pandas are used to implement this project code. The project was implemented on Google Collab.

A. Dataset Description:

The dataset is taken from Kaggle Competition. Parameters like client's previous credits provided by other financial institutions that were reported to Credit Bureau, monthly balances of previous credits in Credit Bureau, repayment history for the previously disbursed credits, etc. are taken into consideration for further predictions.

B. EDA and Feature Engineering:

Exploratory data analysis is applied to check and handle the missing values and necessary data transformations are conducted to process the data. This includes the below mentioned steps:

i. Importing the data:

- Importing the json file from Kaggle which contains the username and the key for the Kaggle account. To link the Kaggle account to Google Collab, creating a new API token on ythe Kaggle account.
- Creating a client by making a directory to host the Kaggle API token.
- Further, use the Kaggle API to import the csv and folders from the data sthece on Kaggle.
- Finally, decompressing these folders to obtain the csv files.

ii. Data Pre-processing:

- Using Pandas, to create data frames for the csv files and extract features which can be used later for analysis.
- Since this data contains some missing values, all the missing values are identified and removed. Later, the unique values of each feature for the train and test dataset is identified.
- In brief, data pre-processing steps will make the dataset ready for training.

```
missing_values = missing_values_table(application_train)
missing_values.head(10)
```

```
Your selected dataframe has 122 columns.
There are 67 columns that have missing values.
```

	Missing Values	% of Total Values
COMMONAREA_MEDI	214865	69.9
COMMONAREA_AVG	214865	69.9
COMMONAREA_MODE	214865	69.9
NONLIVINGAPARTMENTS_MEDI	213514	69.4
NONLIVINGAPARTMENTS_MODE	213514	69.4
NONLIVINGAPARTMENTS_AVG	213514	69.4
FONDKAPREMONT_MODE	210295	68.4
LIVINGAPARTMENTS_MODE	210199	68.4
LIVINGAPARTMENTS_MEDI	210199	68.4
LIVINGAPARTMENTS_AVG	210199	68.4

Figure 1 Removing missing values from training dataset

```
missing_values = missing_values_table(application_test)
missing_values.head(10)
```

```
Your selected dataframe has 121 columns.
There are 64 columns that have missing values.
```

	Missing Values	% of Total Values
COMMONAREA_MODE	33495	68.7
COMMONAREA_MEDI	33495	68.7
COMMONAREA_AVG	33495	68.7
NONLIVINGAPARTMENTS_MEDI	33347	68.4
NONLIVINGAPARTMENTS_AVG	33347	68.4
NONLIVINGAPARTMENTS_MODE	33347	68.4
FONDKAPREMONT_MODE	32797	67.3
LIVINGAPARTMENTS_MODE	32780	67.2
LIVINGAPARTMENTS_MEDI	32780	67.2
LIVINGAPARTMENTS_AVG	32780	67.2

Figure 2 Removing missing values from test dataset

iii. One-hot Encoding:

- Now that we have the processed data, we can train it using One-Hot Encoding.
- What one hot encoding does is, it takes a column which has categorical data, which has been label encoded and then splits the column into multiple columns. The numbers are replaced by 1s and 0s, depending on which column has what value. The basic strategy is to convert each category value into a new column and assign a 1 or 0 (True/False) value to the column. This has the benefit of not weighting a value improperly.
- There are many libraries out there that support one-hot encoding but the simplest one is using **pandas' .get_dummies() method**. This approach is more flexible because it allows encoding as many category columns as you would like and choose how to label the columns

using a prefix. Proper naming will make the rest of the analysis just a little bit easier. This function is named this way because it creates dummy/indicator variables (1 or 0).

- There are mainly three arguments important here, the first one is the Data-Frame you want to encode on, second being the columns argument which lets you specify the columns you want to do encoding on, and third, the prefix argument which lets you specify the prefix for the new columns that will be created after encoding.

```
In [27]: count = 0
         for i in  application_train.loc[:, application_train.dtypes == object]:
            col = application_train[i]
            i = pd.get_dummies(prefix = i, data = col)
            if count == 0:
             data_one_hot = i
            else:
             data_one_hot = pd.concat([data_one_hot,i],axis=1)
            count += 1

         data_one_hot = data_one_hot.drop(columns = ['ORGANIZATION_TYPE_Religion','ORGANIZATION_TYPE_Cleaning','ORG
         ANIZATION_TYPE_Industry: type 8'],axis=1)

         application_train = application_train.drop(columns = application_train.loc[:, application_train.dtypes ==
         object],axis=1)
         application_train = pd.concat([application_train,data_one_hot],axis=1)
         application_train.info()
         application_train.shape

         <class 'pandas.core.frame.DataFrame'>
         Int64Index: 8602 entries, 71 to 307482
         Columns: 232 entries, SK_ID_CURR to EMERGENCYSTATE_MODE_Yes
         dtypes: float64(65), int64(41), uint8(126)
         memory usage: 8.1 MB
Out[27]: (8602, 232)
```

Figure 3 One-hot Encoding

C. Machine Learning:

- In this project, several machine learning models were trained to predict for the loan repayment.
- The machine learning models include: Logistic Regression, Naïve Bayes, KNN (K nearest neighbors), Decision Trees, Random Forest and Gradient Boosting.

- While logistic regression is widely used to solve the classification problem, it sometimes may not be very accurate because it may only capture linear relationships between variables. And the data are not limited to have the linear relationships between variables.
- That is why many different models have also been used to deal with this kind of classification problem.
- To implement all the above mentioned models, the scikit-learn library is imported into the code.

➢ ***Linear Models***

a. *Logistic Regression Model:*

- Logistic regression is a statistical method for predicting binary classes. The outcome or target variable is dichotomous in nature. It describes and estimates the relationship between one dependent binary variable and independent variables.
- A logistic regression model helps us solve, via the Sigmoid function, for situations where the output can take but only two values, 1 or 0. The sigmoid function, also called logistic function gives an 'S' shaped curve that can take any real-valued number and map it into a value between 0 and 1. This process can be used to identify whether or not a customer will be able to repay the loan.
- Here, the model is implemented using the LogisticRegression class imported at the beginning of the code.
- Once the model is defined, it can work to fit the data. Here, the fit method is used on the model to train the data. The fit method takes two parameters here: variables train_final and train_labels.

```
In [33]: # Make the model with the specified regularization parameter
         log_reg = LogisticRegression(class_weight = 'balanced')

         # Train on the training data
         log_reg.fit(train_final, train_labels)

Out[33]: LogisticRegression(C=1.0, class_weight='balanced', dual=False,
                            fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                            max_iter=100, multi_class='auto', n_jobs=None, penalty='l2',
                            random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                            warm_start=False)
```

- Further, the accuracy of the model is checked in order to evaluate its performance. For this, roc_auc_score method of the metrics class is used, as shown below:

Checking The Training Accuracy

```
In [0]: predictions = log_reg.predict(train_final)
        p = pd.Series(predictions)

In [35]: from sklearn.metrics import roc_auc_score

         # Calculate roc auc
         roc_value = roc_auc_score(train_labels, predictions)
         roc_value

Out[35]: 0.5829496211843005
```

- An accuracy of 58% is obtained by this approach.

- *Naïve Bayes:*

- The naive Bayes algorithm reduces the complexity of Bayesian classifier by making an assumption of conditional independence over the training dataset. This drastically reduces the complexity of problem to just 2n.

- The assumption of conditional independence states that, given random variables X, Y and Z, we say X is conditionally independent of Y given Z, if and only if the probability distribution governing X is independent of the value of Y given Z.
- In other words, X and Y are conditionally independent given Z if and only if, given knowledge that Z occurs, knowledge of whether X occurs provides no information on the likelihood of Y occurring, and knowledge of whether Y occurs provides no information on the likelihood of X occurring. This assumption makes the Bayes algorithm, naive.
- In this classifier, the assumption is that data from each label is drawn from a simple Gaussian distribution. The model here is therefore implemented using the GaussianNB class imported at the beginning of the code.
- Once the model is defined, it can work to fit the data. This procedure is implemented in Scikit-Learn's `sklearn.naive_bayes.GaussianNB` estimator. The fit method takes two parameters here: variables train_final and train_labels.

Naive Bayes

```
In [37]: from sklearn.naive_bayes import GaussianNB
         model = GaussianNB(var_smoothing=1e-13)
         model.fit(train_final, train_labels)
Out[37]: GaussianNB(priors=None, var_smoothing=1e-13)
```

- Further, the accuracy of the model is checked in order to evaluate its performance. For this, roc_auc_score method of the metrics class is used, as shown below:

```
In [39]: from sklearn.metrics import roc_auc_score

         # Calculate roc auc
         roc_value = roc_auc_score(train_labels, predictions)
         roc_value
Out[39]: 0.6507800420718007
```

- An accuracy of 65% is obtained by this approach.

➢ Non-Linear Models:

a. *KNN (K nearest neighbors):*

- The k-nearest neighbors (KNN) algorithm is a simple, supervised machine learning algorithm that can be used to solve both classification and regression problems.
- KNN works by finding the distances between a query and all the examples in the data, selecting the specified number examples (K) closest to the query, then votes for the most frequent label (in the case of classification) or averages the labels (in the case of regression).
- The model is implemented using the *KNeighborsClassifier* class imported at the beginning of the code. Brute Force Algorithm is used here to try every possibility to improve efficiency of the model.
- Once the model is defined, it can work to fit the data. Here, *fit* method is used on the model to train the data. The method takes two parameters here: variables train_final and train_labels.

K-Nearest Neighbours

```
In [0]: from sklearn.neighbors import KNeighborsClassifier as KNN

In [42]: neigh = KNN(algorithm = 'brute', n_neighbors=3, leaf_size=30,p=1) #Manhattan Distance
         neigh.fit(train_final, train_labels)

Out[42]: KNeighborsClassifier(algorithm='brute', leaf_size=30, metric='minkowski',
                              metric_params=None, n_jobs=None, n_neighbors=3, p=1,
                              weights='uniform')
```

- Further, the accuracy of the model is checked in order to evaluate its performance. For this, roc_auc_score method of the metrics class is used, as shown below:

```
In [45]: from sklearn.metrics import roc_auc_score
         # Calculate roc auc
         roc_value = roc_auc_score(train_labels, predictions)
         roc_value

Out[45]: 0.5644349214778991
```

- An accuracy of 56% is obtained by this approach.

➢ Decision Trees & Ensemble Learning:

a. *Decision Trees:*

- Tree-based methods can be used for regression or classification. They involve segmenting the prediction space into a number of simple regions. The set of splitting rules can be summarized in a tree, hence the name decision tree methods.
- A decision tree is a flowchart-like tree structure where an internal node represents feature (or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in recursively manner call recursive partitioning.

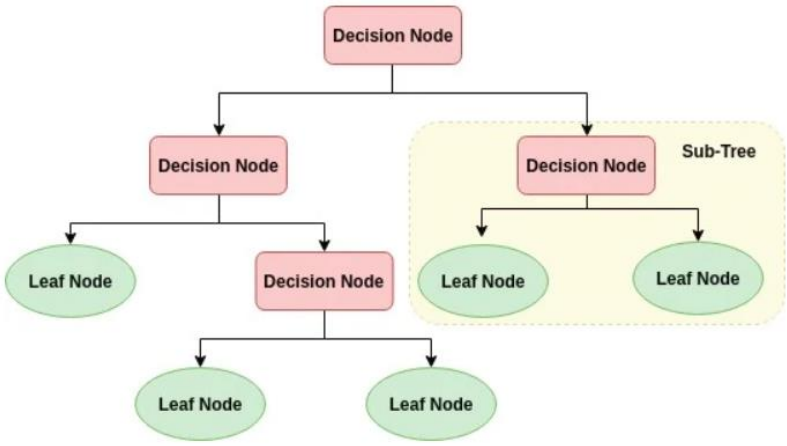

Figure 4 Decision Tree Flowchart

- This flowchart-like structure helps you in decision making. It's visualization like a flowchart diagram which easily mimics the human level thinking. That is why decision trees are easy to understand and interpret.

- Here, the model is implemented using the *DecisionTreeClassifier* imported at the beginning of the code.

Decision Trees

```
In [0]: from sklearn.tree import DecisionTreeClassifier

In [0]: clf = DecisionTreeClassifier()

In [49]: clf.fit(train_final,train_labels)

Out[49]: DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                                max_depth=None, max_features=None, max_leaf_nodes=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, presort='deprecated',
                                random_state=None, splitter='best')
```

b. *Ensemble Learning:*

- *Random Forest:*

- The random forest is a classification algorithm consisting of many decisions trees. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes the model's prediction.
- Random forests are powerful not only in classification/regression but also for purposes such as outlier detection, clustering, and interpreting a data set.

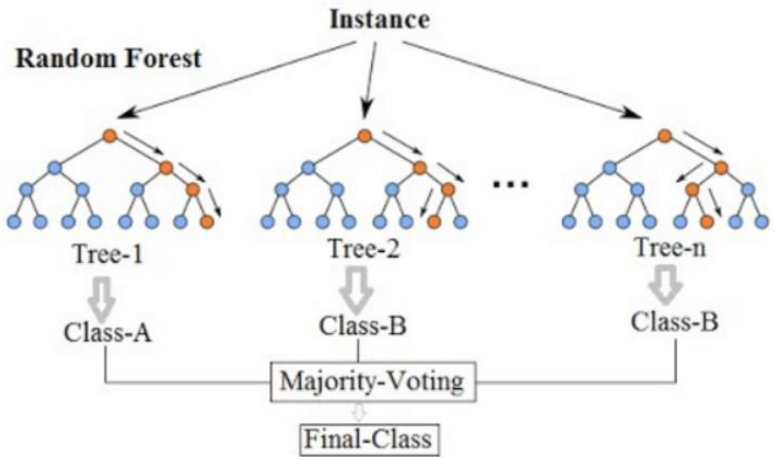

Figure 5 Random Forest Tree Flowchart

- Here, the model is implemented using the *RandomForestRegressor* imported at the beginning of the code. The dataset is then split into training set and test set.
- Once the Random Forest Regression model is defined, it can work to fit the data. Here, *fit* method is used on the model to train the data. The method takes two parameters here: variables train_final and train_labels.

Random Forest

```
In [0]: # Import the model we are using
        from sklearn.ensemble import RandomForestRegressor
        # Instantiate model with 1000 decision trees
        rf = RandomForestRegressor(n_estimators = 5, random_state = 42)
        # Train the model on training data
        rf.fit(train_final, train_labels);
```

- Further, the accuracy of the model is checked in order to evaluate its performance. For this, roc_auc_score method of the metrics class is used, as shown below:

```
In [55]: from sklearn.metrics import roc_auc_score

         # Calculate roc auc
         roc_value = roc_auc_score(train_labels, predictions)
         roc_value
Out[55]: 0.9924661768333908
```

- An accuracy of 99% is obtained by this approach.

- *Gradient Boosting:*

  - Gradient Boosting is an example of boosting algorithm. It is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.
  - The intuition behind gradient boosting algorithm is to repetitively leverage the patterns in residuals and strengthen a model with weak predictions and make it better. Once we reach a stage that residuals do not have any pattern that could be modeled, we can stop modeling residuals (otherwise it might lead to overfitting). Algorithmically, we are minimizing our loss function, such that test loss reach its minima.
  - Gradient Boosting is performed using *GradientBoostingClassfier*.
  - As per below mentioned code snippet, it is observed that after 400th iteration , residuals are randomly distributed around 0 and the predictions are very close to true values. (iterations are called n_estimators in sklearn implementation). This would be a good point to stop else the model will start overfitting.

Gradient Boosting

```
In [0]: from sklearn.ensemble import GradientBoostingClassifier
```

```
In [58]: gb_clf = GradientBoostingClassifier(n_estimators=400, learning_rate=0.7, max_depth=2, random_state=0)
         gb_clf.fit(train_final,train_labels)
```

```
Out[58]: GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                                    learning_rate=0.7, loss='deviance', max_depth=2,
                                    max_features=None, max_leaf_nodes=None,
                                    min_impurity_decrease=0.0, min_impurity_split=None,
                                    min_samples_leaf=1, min_samples_split=2,
                                    min_weight_fraction_leaf=0.0, n_estimators=400,
                                    n_iter_no_change=None, presort='deprecated',
                                    random_state=0, subsample=1.0, tol=0.0001,
                                    validation_fraction=0.1, verbose=0,
                                    warm_start=False)
```

- Further, the accuracy of the model is checked in order to evaluate its performance. For this, roc_auc_score method of the metrics class is used, as shown below:

Checking The Training Accuracy

```
In [0]: predictions = gb_clf.predict(train_final)

In [60]: from sklearn.metrics import roc_auc_score
         # Calculate roc auc
         roc_value = roc_auc_score(train_labels, predictions)
         roc_value

Out[60]: 0.9410027269457266
```

- An accuracy of 94% is obtained by this approach.

D. Results:

- The models that we have created will now be applied to the training and testing data for evaluating the training and validation errors. The goal is to minimize the error with subsequent iterations.
- An ensemble approach minimizes the errors via mechanisms like Bagging and Boosting. Both the procedures are used to handle overfitting, reducing variance and as independent classifiers.
- For this project, Gradient Boosting has been used to achieve this.
- Gradient Boosting trains many models in a gradual, additive and sequential manner.
- GBM identifies the shortcomings of weak learners (eg. decision trees). It identifies the shortcomings by using high weight data points by using gradients in the loss function (y=ax+b+e , where 'e' needs a special mention as it is the error term). The loss function is a measure indicating how good are model's coefficients are at fitting the underlying data.

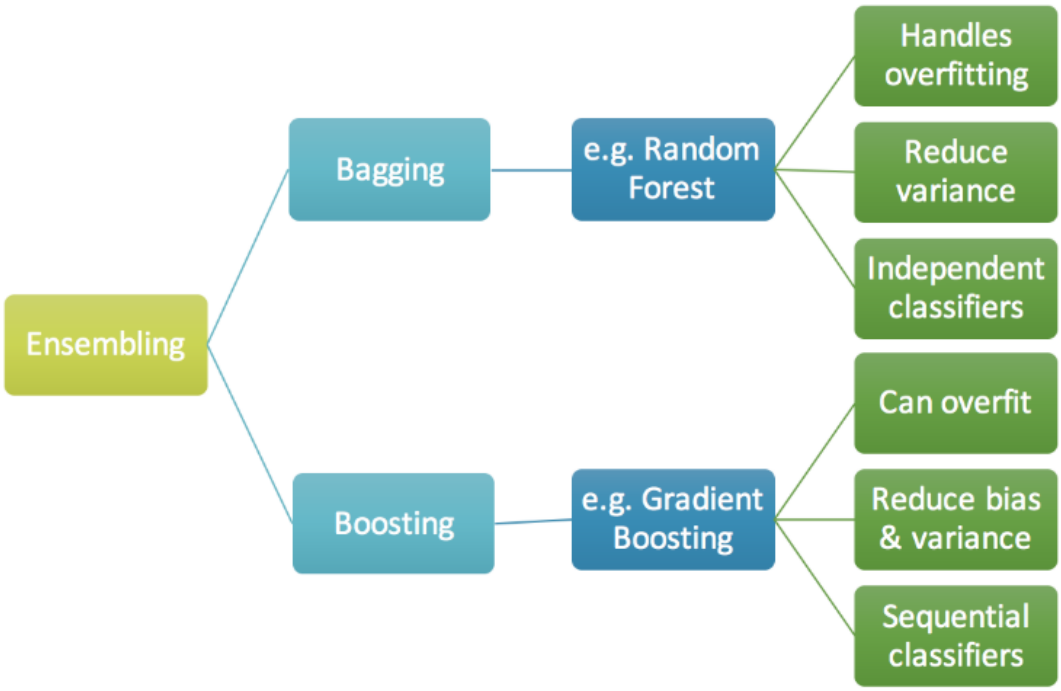

Figure 6 Bagging and Boosting to minimize error

3. CONCLUSION AND FUTURE WORKS:

The loan business has always been a popular trend, and many people apply for loans for various reasons. However, there are cases where people do not repay the bulk of the loan amount to the bank which results in huge financial loss. Hence, if there is a way that can efficiently classify the loaners in advance, it would greatly prevent the financial loss to many financial institutions.

In this project, to achieve the best performance, the dataset was cleaned first, and the exploratory data analysis and feature engineering were performed. The strategies to deal with both missing values and imbalanced datasets were covered. Further, five machine learning models which are Logistic Regression, Naïve Bayes, KNN (K nearest neighbors), Decision Trees, Random Forest and Gradient Boosting are applied to predict if the applicant could repay the loan.

After training the models, it is noticed that the model found best for the dataset with highest accuracy is the Random Forest model. Moreover, with the application of Gradient Boosting mechanism, better results were achieved for the prediction.

In the report, we demonstrated the use of machine learning algorithms on a very challenging dataset to predict loan repayment ability. More sophisticated learning algorithms and dimension reduction techniques could be used in future to further improve model performance on this important prediction task.

4. REFERENCES:

[1] “Home Credit Default Risk.” Kaggle, https://www.kaggle.com/c/homecredit-default-risk/data.

[2] Gorton, Gary, and James Kahn. ”The design of bank loan contracts.” The Review of Financial Studies 13, no. 2 (2000): 331-364.

[3] Langrehr, Virginia B., and Frederick W. Langrehr. ”Measuring the ability to repay: The residual income ratio.” Journal of Consumer Affairs 23, no. 2 (1989): 393-406

[4] Kolo, Brian, Thomas Rickett McGraw, and Dathan Gaskill. ”Systems and methods for using data metrics for credit score analysis.” U.S. Patent Application 13/456,532, filed November 1, 2012.

[5] C¸ elik, S¸ aban. ”Micro credit risk metrics: a comprehensive review.” Intelligent Systems in Accounting, Finance and Management 20, no. 4 (2013): 233-272.

[6] Olivas, Michael A. ”Paying for a law degree: Trends in student borrowing and the ability to repay debt.” J. Legal Educ. 49 (1999): 333.

[7] Hesseldenz, Jon, and David Stockham. ”National direct student loan defaulters: The ability to repay.” Research in Higher Education 17, no. 1 (1982): 3-14.

[8] Flint, Thomas A. ”Predicting student loan defaults.” The Journal of Higher Education 68, no. 3 (1997): 322-354.

[9] Afolabi, J. A. ”Analysis of loan repayment among small scale farmers in Oyo State, Nigeria.” Journal of Social Sciences 22, no. 2 (2010): 115-119.

[10] Wongnaa, C. A., and Dadson Awunyo-Vitor. ”Factors affecting loan repayment performance among yam farmers in the Sene District, Ghana.” Agris on-line Papers in Economics and Informatics 5, no. 665-2016-44943 (2013): 111-122.

[11] Murdock, C.W., 2011. The Dodd-Frank Wall Street Reform and Consumer Protection Act: What Caused the Financial Crisis and Will Dodd-Frank Prevent Future Crises. SMUL Rev., 64, p.1243.

[12] Ivashina, Victoria, and David Scharfstein. ”Bank lending during the financial crisis of 2008.” Journal of Financial economics 97, no. 3 (2010): 319-338.

[13] Mierzewski, Michael B., Christopher L. Allen, Jeremy W. Hochberg, and Kevin Hall. ”CFPB Finalizes Ability-to-Repay and Qualified Mortgage Rule.” Banking LJ 130 (2013): 611.

[14] Liaw, Andy, and Matthew Wiener. ”Classification and regression by randomForest.” R news 2.3 (2002): 18-22.

[15] Ke, Guolin, et al. ”Lightgbm: A highly efficient gradient boosting decision tree.” Advances in Neural Information Processing Systems. 2017.

[16] Chawla, Nitesh V., et al. ”SMOTE: synthetic minority over-sampling technique.” Journal of artificial intelligence research 16 (2002): 321- 357.

[17] Arthur, David, and Sergei Vassilvitskii. ”k-means++: The advantages of careful seeding.” Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 2007.

[18] Laurens van der Maaten, Geoffrey Hinton. ”Visualizing Data using t-SNE.” Journal of Machine Learning Research, 2008.

[19] Visualising High-dimensional Datasets Using PCA and t-SNE. https://towardsdatascience.com/visualising-high-dimensional-datasetsusing-pca-and-t-sne-in-python