ERASMUS SCHOOL OF ECONOMICS
BACHELOR THESIS - ECONOMETRICS AND OPERATIONS RESEARCH

Risk Assessment in Peer-to-Peer Lending using Machine Learning Methods

Author

PRAKASH BHAGAT (495719)

Supervisor

UTKU KARACA

Second Assessor

OKTAY KARABAG

Abstract

This paper discusses several popular methods used for the classification of borrowers in Peer-to-Peer (P2P) lending. Using data obtained from the P2P lending platform - Social Lending, the work of Malekipirbazari and Aksakalli (2015) is replicated and extended. The dataset contains 18 features regarding borrowers and as the majority of borrowers did not default on their loan a cost sensitive method was implemented to combat this. The classifiers used are K Nearest Neighbours, Logistic Regression, Support Vector Machines, Random Forests, and XGBoost. These classifiers are explained comprehensively, discussing how they function as well as explaining both their advantages and limitations. The results are in line with the work of Malekipirbazari and Aksakalli (2015), finding that Random Forests outperform the other classifiers in classifying good borrowers. As an extension to these results, a focus is placed on boosting and bagging methods, where Random Forests outperform the results of XGBoost by a large margin.

Contents

1 Introduction

1.1 Research Problem

Peer-to-peer (P2P) lending originated in 2005 and is still relatively new in the research field. Such platforms remove the need for a bank and allow lenders and borrowers to interact directly using an auction process where lenders offer loans. The lender offers the amount to loan and the minimum interest they are willing to accept. Other lenders may then undercut these offers by offering lower interest rates. Typically, the interest rates from lending is higher than the amount offered at a bank due to the increased risk of a borrower defaulting. To ensure lenders are offering loans to creditable individuals, borrowers are required to provide various financial statistics and demographic characteristics. The most commonly used financial statistics are FICO scores, debt-to-income ratio, monthly income, and homeownership. A more detailed description of the process of peer-to-peer lending can be found in the paper by Bachmann et al. (2011). In many cases, the determining factor of whether a borrower can receive a loan is the FICO score, however, this is often a misleading statistic. Malekipirbazari and Aksakalli (2015) found accepting individuals with a FICO score above 750 resulted in an acceptance rate of 8%. The default rate of these borrowers was 8.2%. Implementing the same technique with Random Forests and accepting only 8% of the borrowers led to a default rate of only 3.1%. This paper replicates and extends their work, applying machine learning techniques to improve the classification of good borrowers in P2P lending. There are a wide variety of techniques to choose from, and ensemble methods have shown great success in the past Lessmann, Baesens, Seow and Thomas (2015). In particular, bagging algorithms such as Random Forests often provide consistently accurate results. Extreme Gradient Boosting or XGBoost (Chen & Guestrin, 2016) is a boosting algorithm that has demonstrated impressive results in classification problems often outperforming many other standard classifiers. Thus, the following research question is posed:

Does XGBoost outperform Random Forests when assessing risk in P2P lending?

The results of XGBoost are compared to other widely used classifiers namely, Logistic Regression, Support Vector Machines, and K-Nearest Neighbours. Additionally, to optimize the performance of Random Forests and XGBoost, several hyperparameters are investigated and tuned to optimize performance. To answer this research question, commonly used performance metrics such as Accuracy, AUC, and the F1-score are used. We thus also evaluate whether the results change significantly when different metrics are used.

1.2 Motivation

Credit risk assessment using machine learning has been covered extensively in the past, however, the amount of research in assessing risk in the format of peer-to-peer lending is far less. These P2P lending platforms use a proprietary grading system when evaluating potential borrowers similar to how banks use FICO scores when their clients apply for loans. Malekipirbazari and Aksakalli (2015) found better results when using Random Forest compared to the grading system from Lending Club. This paper, however, came out before the publication and release of XGBoost. Thus, as an extension, we test how XGBoost compares to the classification methods used in this paper and specifically how it performs against the best method - Random Forests. For a relatively new algorithm, XGBoost has performed very well in many machine learning competitions on Kaggle. For example, in a 2015 Kaggle classification competition, of the 29 winning solutions 17 used XGBoost. Additionally, bagging and boosting algorithms are among the most used ensemble classifiers which are often compared in classification problems Maclin and Opitz (1997). This paper extends this discussion and finds which method is best for this dataset.

1.3 Relevance

Thoroughly understanding why different algorithms perform better than others is of great importance to businesses to know which to implement and in what situation. In this paper, the properties of XGBoost are described in detail and thus provides the reader with an insight into how the method works and how it differs from other boosting algorithms. Additionally, the major differences between bagging and boosting algorithms is discussed. With the recent success XGBoost has had, this paper evaluates how the results compare and whether it is in line with previous results. As XGBoost is a new method there is not much literature on it in the case of risk assessment. By utilizing this method, we assess whether it provides better results than standard classifiers and how large the differences are. In the case that it offers significantly better results this provides valuable insights to companies such as insurance firms when assessing the risk of potential clients.

The paper is structured as follows: Section 2 discusses previous literature on media mix machine learning algorithms in the context of risk assessment. Thereafter, Section 3 describes the characteristics of the data set as well the pre-processing methods used. Following this, the Methodology will be represented in Section 4 and the results are discussed in Section 5. Finally, the answer to the research question is addressed in Section 6, as well as the main findings.

2 Literature Review

There is extensive research on the performance of machine learning techniques in risk assessment as previously mentioned. In this section, the main findings of such papers are discussed. Malekipirbazari and Aksakalli (2015) compared the results of Logistic Regression, K-Nearest Neighbours, Support Vector Machines (SVM), and Random Forests. Random Forests were found to perform best with an accuracy of 78% with the next best model being K-Nearest Neighbours achieving an accuracy of 70.1%. Using the same classifiers and data, this paper investigates whether the results are significantly different. These results are in line with those found by Lessmann et al. (2015) who found ensemble models produce more accurate predictions than single classifiers. Single classifiers such as Logistic Regression, an industry-standard model, or SVM, estimate outcomes to develop a single classification model. Contrastingly, ensemble classifiers typically use a weighted average of multiple base models in their evaluation. In Figure 1, the differences between single and ensemble classifiers are shown clearly. Additionally, Kim, Min and Han (2006) found that ensemble models were able to reduce the bias and variance compared to single classifiers.

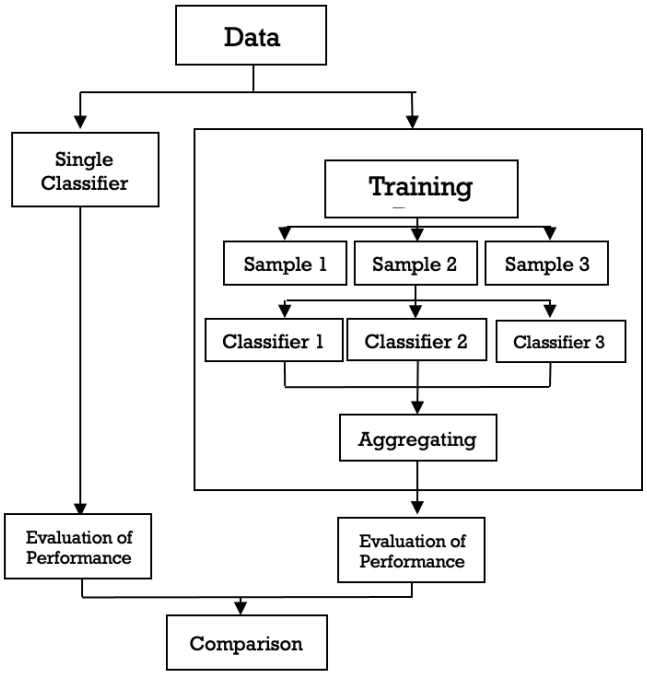


Figure 1: Workflow of Single and Ensemble Classifiers

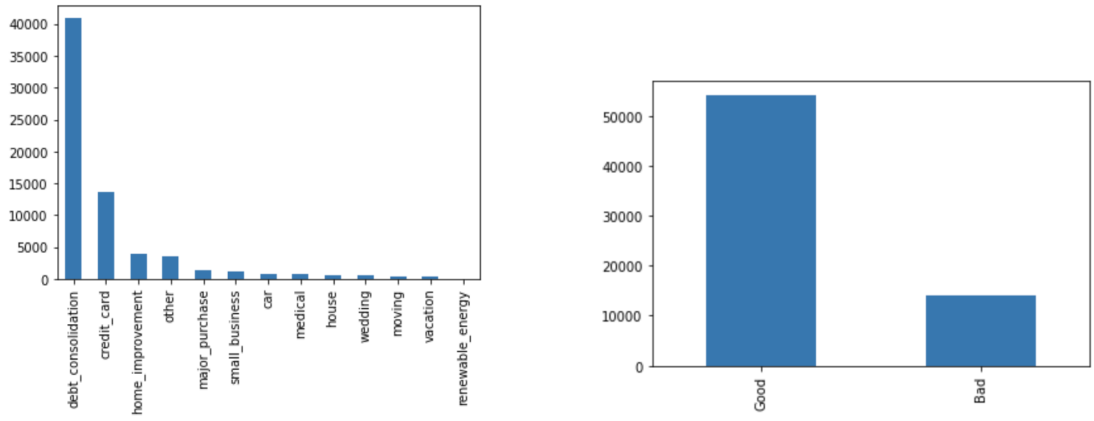
The most commonly used ensemble models are bagging (Breiman, 1996) and boosting (Freund, Schapire et al., 1996) algorithms. The idea behind bagging and boosting algorithms is to modify weak learning techniques to improve performance. A weak learning algorithm provides results slightly better than random chance. In this case, decision trees are used which often have poor predictive performance due to large variances. Bagging algorithms such as Random Forests create independent base models from a subset of the data and take a weighted average of all predictions. Boosting, similarly to bagging, averages the results of decision trees however, in contrast to bagging, create additive models which learn from the errors in previous trees. Specifically, at each tree, the algorithm evaluates how well the model is performing and reweights the data focusing on areas that are not performing well. In this way, it works more efficiently than bagging and often outperforms it. The research question proposed argues whether this statement is true or false for this specific dataset.

Though XGBoost is still relatively new, there are a few articles that compare its performance to other classifiers. Fan et al. (2018) compared the prediction results between SVM and XGBoost for the prediction of solar radiation. While the results were found to be similar in terms of prediction accuracy, XGBoost was found to be more stable and efficient with its running time. Given the large dataset investigated, this will be useful in reducing running times. Nielsen (2016) provided several reasons why tree boosting algorithms perform so well. They found as boosting algorithms create additive models they are more robust in high dimensional datasets. The paper also mentions several reasons why XGBoost has outperformed other gradient boosting methods. By including an extra parameter that penalizes individual trees and with the addition of an extra randomization parameter, XGBoost can improve the bias-variance trade-off significantly compared to other methods.

3 Data

The data used is taken from The Lending Club over the period September 2012 - September 2014. A cleaned version of the data provided by Malekipirbazari and Aksakalli (2015) contains 68252 observations which has removed loans that hadn't been either defaulted or accepted. The original data from Lending Club required borrowers to provide information across 35 categories. Malekipirbazari and Aksakalli (2015) selected 23 of these features and developed a predictive model using a total of 15 features. Of these 15, several were created by the authors, taking ratios of original variables. The dependent variable is the borrower status where *Good* indicates that the loan has been fully paid off and *Bad* means the borrower defaulted. Excluding the dependant variable, there are 16

other features of which 3 are categorical. A description of these variables can be found in Table 7 in the Appendix.



(a) Purpose (b) Loan Status

Figure 2: Histograms for Purpose and Loan Status

Figure 2 shows a few histograms providing some insight into the characteristics of the features. The most notable insights are that the majority of borrowers repaid their loans in full and the purpose of the loan in many cases was used for debt consolidation. Figure 8 in the Appendix, shows several other histograms where we see that the average FICO score is 675 and the mean LC Grade is approximately 24. From Figure (d) we also see that the loan amount ranged from $1000 to $35000 and a mean of near $14000. The categorical features were transformed to numerical variables using one-hot encoding except when implementing Random Forests which can handle categorical inputs. The *RevolvingUtilizationRate* has 64 missing values which are filled using the mean value. Subsequently, the correlation between the features and dependant variable is calculated which is shown in Table 1. Many of the individual attributes are weakly correlated to the dependant variable with the highest correlation of only 0.198 when using the *LCGrade*. Note that in the models, the variables *FICO* and *LCGrade* are excluded to match the results in Malekipirbazari and Aksakalli (2015).

4 Methodology

In this Section, the classifiers used are explained as well as the motivation for using them. Two methods of cross-validation are used. Firstly, k=5 folds cross-validation, and secondly, stratified train-test-split. Once the classification methods have been implemented, the hyperparameters of XGBoost and Random Forests are investigated. Hyperparamter tuning is attempted, however due to

Table 1: Correlation to loan status

Rank	Attribute	Correlation
1	LC Grade	0.198
2	Income to payment ratio	0.143
3	Annual income	0.126
4	FICO Score	0.125
5	DTI	-0.119
6	Revolving line utilized	-0.111
7	Term	-0.104
8	Revolving to income ratio	0.079
9	Home ownership: mortgage	0.075
10	Total accounts	0.073
11	Home ownership: rent	-0.072
12	Loan Purpose: small business	-0.054
13	Credit age	0.044
14	Loan amount	-0.044
15	Inquiries	0.002

time constraints and restrictions with storage this failed to complete. Finally, to determine which classifier performs best, several performance metrics are used. Namely, these are the Accuracy, AUC, RMSE, F1 score, True Positive and False Positive Rates. Firstly, we replicate the methods by Malekipirbazari and Aksakalli (2015) whereby all techniques are implemented in WEKA (Witten & Frank, 2005), a statistical software used for machine learning problems. To account for the imbalanced ratio between classes, a weighting function of 5:1 is used where the cost of identifying a *Bad* borrower as *Good* is 5 times worse than classifying a *Good* borrower as *Bad*. This is done using the $costsensitiveclassifier$ function within WEKA. Additionally, the results for XGBoost are replicated in Python. Instead of using cost sensitivity, random oversampling is used to balance the number of instances between classes to improve the performance of the classifiers. Additionally, a stratified train-test split is used as opposed to k-folds cross-validation which maintains the same ratio of Good and Bad borrowers between training and test sets.

4.1 Logistic Regression

Logistic Regression is an industry standard for classification and prediction problems (Lessmann et al., 2015) and offers reliable results. It is simple to implement and often used as the benchmark when evaluating how classifiers perform. It is used in problems to predict the conditional probability of a sample belonging to a certain class. Using the sigmoid function, it transforms linear regression into the logit function.

$$p(X) = \frac{e^{X'\beta}}{1 + e^{X'\beta}} \tag{1}$$

The equation above represents the logistic function where X' is the transpose of the column vector of explanatory variables which is a 1 x h vector. β is an h x 1 column vector where h represents the total number of explanatory variables used. This equation can be manipulated into the following form:

$$\frac{p(X)}{1-p(X)} = e^{X'\beta} \tag{2}$$

The left side of this equation represents the Odds Ratio. This a ratio between the probability of having a *Good* borrower and the probability of getting a *Bad* borrower. Taking the log of both sides results in Equation 3:

$$log[\frac{p(X)}{1-p(X)}] = X'\beta = x_1\beta_1 + x_2\beta_2 + ... + x_h\beta_h \tag{3}$$

$log[p(X)/1-p(X)]$ is the log-odds or logit value. Increasing X by one unit results in an increase of β in the log-odds value. Next, the coefficients β_1, β_2 ... β_h within the *beta* vector are estimated such that the β values chosen maximize the following likelihood function:

$$l(\beta) = \prod_{y=1} p(X) \prod_{y=0} (1-p(X)) \tag{4}$$

Once the coefficients have been estimated, the probability of default is calculated using Equation 1 for each instance.

Several assumptions are needed for Logistic Regression to perform well: Firstly, the dependant variable must be categorical. Secondly, the independent variables should not be highly correlated with each other. Figure 9 in the Appendix, shows a heatmap where the values indicate the correlation between variables. This shows that a few variables are notable negatively correlated to each other. This issue is discussed in more detail in Section 5. Finally, the number of instances should be large as otherwise, the parameter estimates are often unstable. This is not an issue as the dataset is relatively large.

4.2 K Nearest Neighbours

K-nearest neighbours (KNN)(Cover & Hart, 1967) is a supervised learning algorithm used in classification and regression problems. It uses a simple method and produces good results. For a binary classification problem, each point uses K points closest to it to identify which class it belongs to. This is measured using a method to calculate the distance which is often the Euclidean distance. For example, if K=5 and 4 of these points belong to class 1 and the last point belongs to class 2, that instance would be classified as belonging to class 1. The number of neighbours chosen is

critical to the performance as small values can lead to overfitting while too large a value reduces the effectiveness. For simplicity, K=1 is chosen. Figure 3 illustrates the effect K has on the accuracy and amount of overfitting. This shows the amount of overfitting is very large at first and reduces significantly. Using a value of K greater than 10 results in very similar results between the training and test sets.

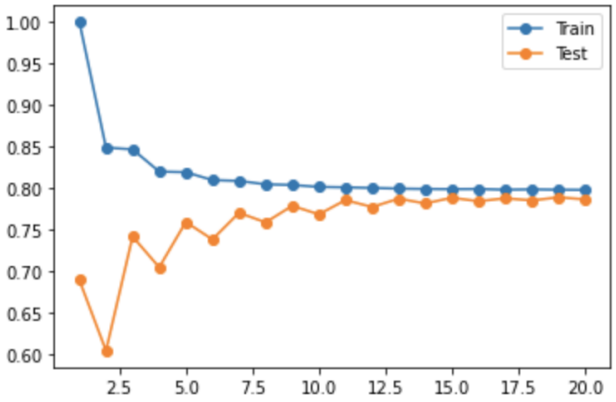


Figure 3: Plot of Accuracy against K value for KNN

4.3 Support Vector Machines

Support Vector Machines (SVM) (Cortes & Vapnik, 1995) is another popular supervised machine learning algorithm often used in binary classification. The main idea for a binary problem is to create a hyperplane, creating a line between the two classes $y_i \in [-1, 1]$. There are p training pairs $(x_1, y_1)...(x_p, y_p)$, $x \in \mathrm{R}^d$ where d is the dimension of the vector. In the explanation below d=2 is considered to simplify terms. The points in either class closest to this line, are known as support vectors and the algorithm aims to have the hyper-plane as far away from the support vectors as possible. A visual representation of this is shown in Figure 4 for the separable and non-separable case taken from Burges (1998).

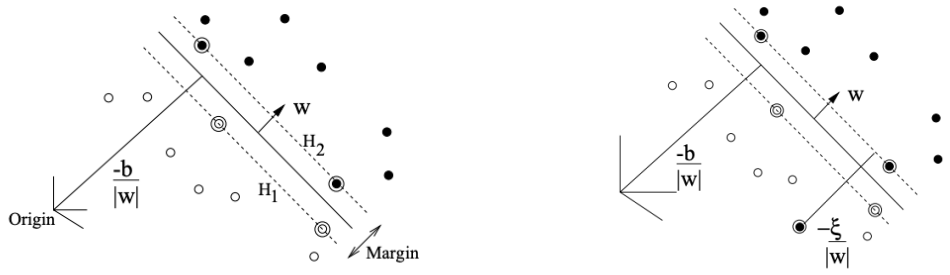
Given a hyper-plane:

$$y = a * x + b \tag{5}$$

$$0 = a * x + b - y \tag{6}$$

and defining the vectors X=(x,y) and w = (a,-1), the hyper-plane in vector form becomes:

$$w \cdot X + b = 0 \tag{7}$$



(a) Separable Data (b) Non-Separable Data

Figure 4: Support Vectors for Separable and non-separable data

The distance between this hyper-plane and H1 and H2 is known as the margin. The points lying on H1 and H2 are the support vectors. The aim of SVM is to maximize the margin such that the classifier can identify the class of new data points. The SVM problem is formualated as follows:

$$w.x_i + b \geq 1 \ \ for \ \ y_i = +1 \tag{8}$$

$$w.x_i + b \leq -1 \ \ for \ \ y_i = -1 \tag{9}$$

Combining these equations, this can be written as:

$$y_i(w.x_i + b) - 1 \geq 0 \ \ for \ \ y_i = +1, -1 \tag{10}$$

The points lying on H1 thus satisfy the equation: $w.x+b$ = -1, and the points on H2 satisfy $w.x+b$ = +1. The distance between H1 and the origin is (-1-b)/$|w|$ and similarly, the distance between H2 and the origin is (1-b)/$|w|$. The margin is the difference between these distances and is equal to $1/|w|$. The optimization now becomes:

$$max \frac{1}{|w|} = min|w| \tag{11}$$

This can be rewritten such that the general SVM optimization problem is formulated as:

$$min_{w,b} \ \ \frac{1}{2}||w||^2 \tag{12}$$

$$subject\ to\ y_i(x_i \cdot \mathbf{w} + b) - 1 \geq 0,\ \ i = 1, ..., m \tag{13}$$

To solve this, the Lagrangian function is needed as well as the Lagrangian multiplier a_i for each constraint function:

$$L(w, b, a) = \frac{1}{2}||w||^2 - \sum_{i=1}^{m} a_i[y_i(x_i \cdot \mathbf{w} + b) - 1] \tag{14}$$

The solution to the primal problem is found by solving the following Lagrangian problem:

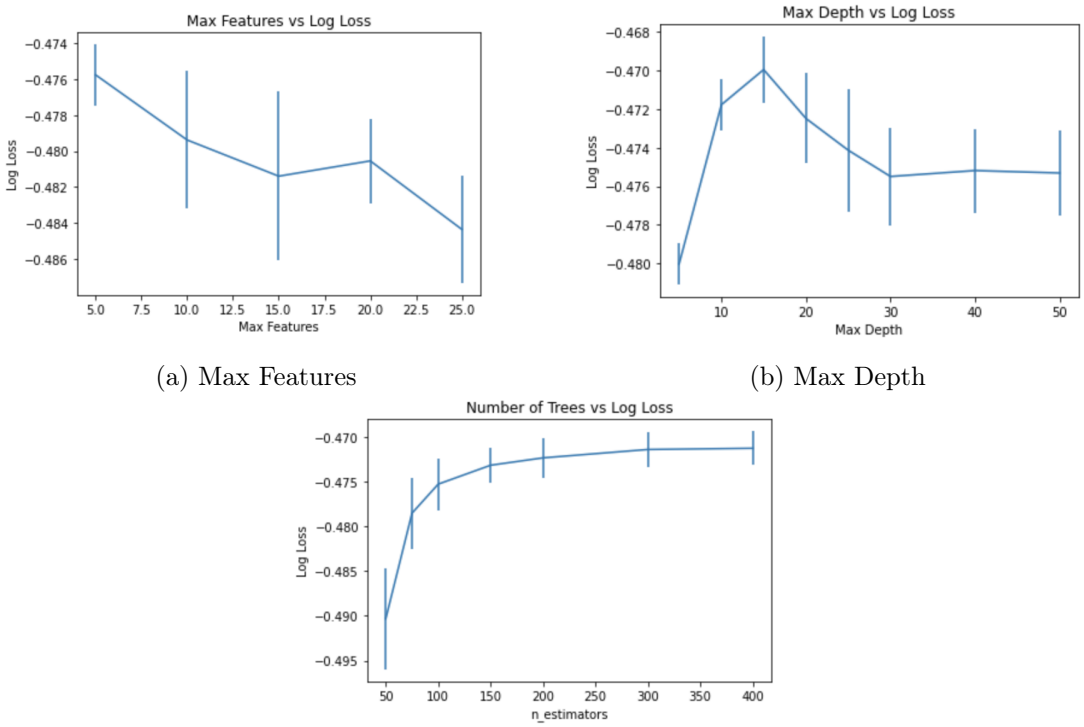
$$min_{w,b}\ max_a L(w, b, a) \tag{15}$$

$$subject\ to\ a_i \geq 0\ \ for\ \ i = 1, ..., m \tag{16}$$

With imbalanced data, the hyper-plane is shifted to the majority class resulting in misrepresentative results. In most cases, the data is not separable in a 2-dimensional space, and to solve this SVM uses a kernel function. A kernel function converts data in low dimensional space and transforms it into a higher dimension. SVM also has a cost-sensitivity parameter called the Regularization which asks the system what the cost of misclassifying an instance is. There are many other parameters however changing the kernel and regularization parameter have the largest effect. The running time for SVM is very time-consuming and computationally expensive and thus tuning of these parameters is not performed.

4.4 Random Forests

Random Forests (Breiman, 2001) is a bagging algorithm that trains decision trees as base learners in a parallel environment. Each node in a decision tree represents a test for which the branch of that node indicates the result of the test. In each tree, a prediction is made using a random subset b of features which improves the bias-variance trade-off. A new sample of b predictors is taken at each split where b is usually set to $\sqrt{p}$, where p is the total number of predictors. This reduces the correlation between trees which increases the performance compared to other bagging classifiers which take b=p. Each tree starts with a root node and branches out until a maximum depth that is pre-specified is reached. At each node, the feature chosen from the subset is selected by using the Gini Impurity. This value calculates the probability that a randomly chosen sample in the node is misclassified using the features in that node. Thus at each node, the feature chosen to split on is the one that results in the greatest reduction in the Gini Impurity. Once the specified number of trees has been reached the algorithm takes the average of all trees to give an overall prediction.



(a) Max Features

(b) Max Depth

(c) Number of Trees

Figure 5: Max Features, Max Depth, Number of Trees against Log Loss for Random Forest

Many features can be tuned within Random Forests to improve predictive performance namely, the number of trees built, the maximum number of features used when splitting a node, and the maximum depth of each tree. Increasing the number of trees improves the performance up to an extent after which the computational cost outweighs the benefit. Increasing the number of features also improves performance however increases the correlation between trees. The ideal number achieves the best trade-off between performance and correlation. Similarly, this is the same case when increasing the max depth of each tree. Figure 5 shows the change in Log Loss when adjusting these parameters. Based on this, the optimal values to use are 5 for max features, 15 for max depth, and 400 trees. Having said this, as the improvement in log loss is marginal when increasing the number of trees, this is set to 80 and max depth is set to 25 as done in Malekipirbazari and Aksakalli (2015). Random Forests offer many advantages over other classifiers. For example, it can handle categorical values unlike many other classifiers and performs implicit feature selection. Furthermore, it is very quick to train and hyperparameter tuning is much faster than it is for SVM or Gradient Boosting Machines (GBM) as there are fewer features to adjust. Lastly, when comparing it to GBM's such as XGBoost, it is more robust to overfitting as it trains trees on different subsamples of the data. The pseudo-code for Random Forests is shown in Algorithm 1 in the Appendix.

4.5 XGBoost

XGBoost is a subset of gradient boosting machines (GBM). Gradient boosting techniques are additive models which use decision trees as weak learners where trees are built in sequence to reduce the errors of previous trees. Firstly, a model is created using decision trees to fit a model to the data. Next, the residuals are calculated and a model is fitted on these residuals which becomes a boosted version of the first model. In each step, the previous model remains unaltered and the models fitted on the residuals are used to reduce the errors. By doing so, this also reduces the amount of overfitting of the model. The initial model $F_0(x)$ is defined to minimize the loss function which can be any differentiable function.

$$F_0(x) = arg\,min_\gamma \sum_{i=1}^{n} L(y_i, \gamma) \tag{17}$$

The first step is creating a constant model such as $F_0(x) = \gamma$. The residual from this is then taken by subtracting the prediction from the observed value. As we are building a classification model the log(odds) ratio is needed. For example, if 80% of individuals are good borrowers and the remaining 20% are bad, the odds ratio is 4:1. In the first step, shown by Equation 17, we need to find a log(odds) value that minimizes the loss function. This loss function can be any differentiable function and the negative log-likelihood is often used which is shown below:

$$L(y_i, p_i) = -[y_i log(p_i) + (1 - yi) log(1 - pi)] \tag{18}$$

y_i are the observed values which can be either 0 or 1, p_i is the predicted value which is between 0 and 1 and γ represents a log(odds) value. Next, the pseudo residuals are calculated where i is the sample number and m is the tree built. This is done as follows:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1(x)}} for \quad i = 1...n \tag{19}$$

This is the partial derivative with respect to the predicted log(odds): $F(x_i)$. In other words, the gradient of the loss function. The loss function uses the observed values y_i and the latest predictions $F(x_i)$. $F(x) = F_{m-1(x)}$ simply indicates that the most recent predicted log odds value is used. The pseudo residuals for each iteration are then used to fit a regression tree represented by $h_m(x_i)$. Subsequently, the output values γ_m are calculated for each leaf shown in Equation 20:

$$\gamma_m = arg\,min_\gamma \sum_{x_i=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)) \tag{20}$$

This is similar to the initial equation, where the loss function is minimized based on the log(odds) value gamma. Finally, a new prediction is made for a new sample. This prediction is based on the previous prediction $F_0(x)$ and the output values from the first tree made.

$$F_1(x) = F_0(x) + \gamma_1 h_1(x) \tag{21}$$

This is the prediction for the first tree. The process then restarts and for m=2 up to m=M, where M is the number of base learners used. This value is pre-specified by the user. In this case, a value of M=50 is used, which is explained later in this section. The Figure below below represents how gradient descent algorithms work. Additionally, Algorithm 2 in the Appendix shows the pseudo-code for GBM.

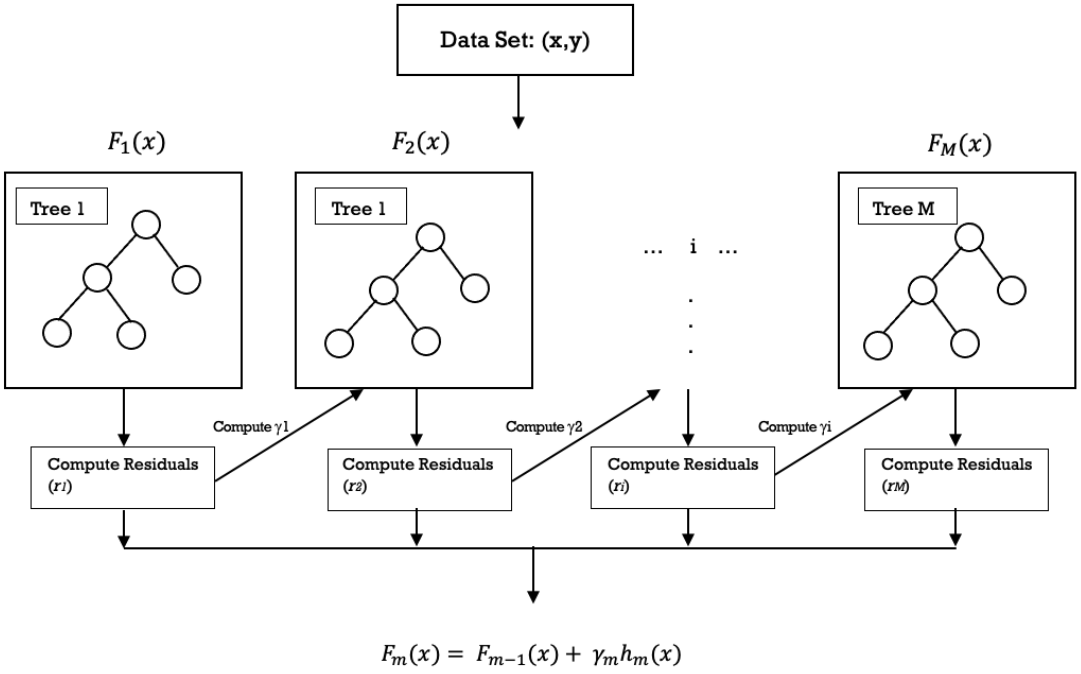


Figure 6: GBM Workflow

XGBoost is similar to GBM in that it also uses gradient descent in minimizing the loss function, as well as including the same base set of hyperparameters, however, the loss function used is adjusted

to control for the complexity of the trees.

$$L_{xgb} = \sum_{i=1}^{N} L(y_i, F(x_i)) + \sum_{m=1}^{M} \Omega(hm) \tag{22}$$

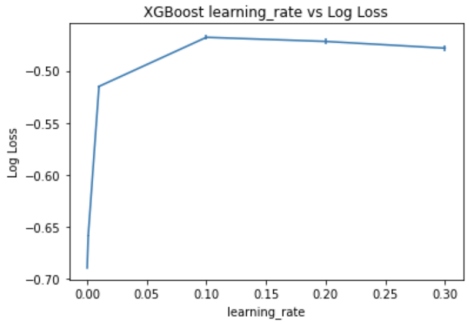
$$\Omega(h) = \gamma T + \frac{1}{2}\lambda ||w||^2 \tag{23}$$

where T represents the number of leaves of the tree and w are the output scores for the leaves. γ is the minimum loss reduction needed to split a node and higher values of γ lead to simpler trees. Once a tree is grown to a pre-determined level this parameter works backward removing less informative leaf nodes. This results in fewer, more optimized trees plays and is one of the keys to its high success rate. XGBoost uses L1 and L2 regularization to reduce the amount of overfitting. Essentially, regularization reduces noise caused by the explanatory variables. L1 and L2 regularization is used in Lasso and Ridge regression respectively and are commonly used techniques for large datasets where there is a significant amount of overfitting. Finally, XGBoost can train the data quicker than other methods and makes the most use of the processing power of the machine it is run on.

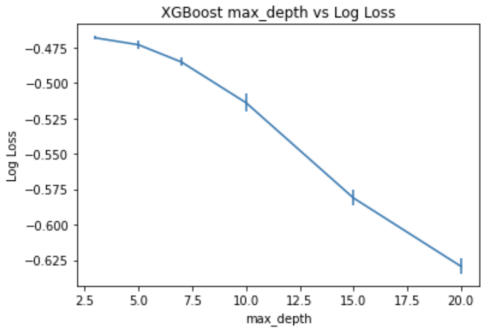
The R Plugin is used within WEKA to implement XGBoost. Hyperparameter tuning was very time-consuming in both WEKA and Python and thus not all parameters were tuned. To improve performance as much as possible, the learning rate, number of trees, and the maximum depth of each tree were adjusted as these factors affect the performance of XGBoost the most. Increasing the number of trees improves the average results, however, lengthens the running time of the classifier. The max depth of each tree represents the maximum number of features used in each tree where too high a value can lead to overfitting. Finally, the learning rate applies a weighting factor made by new trees, for example, a learning rate of less than one makes fewer corrections for each tree in the model thus more trees are needed. Figure 7 shows that for this dataset a high learning rate of 0.1 performs best. Figure (b) shows that according to log loss shallower trees perform better thus a value of 5 is used. Finally, Figure (c) shows that as the number of trees is increased the logloss value gets marginally worse and so a value of 50 trees is used.

4.6 Performance Metrics

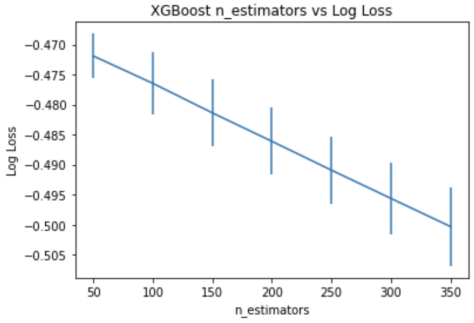
At each prediction, multiple performance metrics are used. Firstly, the overall accuracy of each fold is calculated as a benchmark grade. This, however, does not provide a proper indication of model performance for imbalanced data. For example, classifying all instances as *Good* results in



(a) Learning Rate



(b) Max Depth



(c) Number of Trees

Figure 7: Max Features, Max Depth, Number of Trees against Log Loss for XGBoost

an accuracy of 80% which is quite misleading. Thus we focus on alternative metrics the many constructed using the confusion matrix, shown in Table 2.

Table 2: Confusion Matrix

	Positive Predictions	Negative Predictions
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Firstly, the TP and FP Rates are calculated for *Good* and *Bad* borrowers. These are calculated as follows:

$$TP\ Rate = Recall = \frac{TP}{TP + FN} \quad FP\ Rate = \frac{FP}{FP + TN} \tag{24}$$

where the FP Rate indicates the misclassification rate. Next, the precision and recall are calculated needed to calculate the F1-score. This metric is widely used to indicate overall performance and is the harmonic mean of Precision and Recall. Additionally, the Receiver-Operating-characteristic is used which illustrates the relation between Specificity and Recall. Recall shows how well the positive class is predicted whereas the Specificity denotes how well the negative class is predicted.

The ROC curve plots the TP on the y-axis and FP on the x-axis, thus the ideal point being (0,1). Comparison of multiple classifiers using the ROC curve can be cumbersome thus the classifiers will be compared using the Area under Curve (AUC). Each classifier is compared by the area under the ROC curve and the value lays between 0 and 1. A perfect skill classifier has an AUC of 1, thus the higher the value of the AUC the better the model is at predicting the positive class. In general, an AUC of 0.5 indicates poor performance while a value between 0.8 and 0.9 indicates excellent predictive performance. Finally, the Root-Mean-Squared-Error (RMSE) is calculated which is a good measure of accuracy to compare prediction errors of different models, where lower values indicate a better fit.

5 Results

5.1 Results from WEKA

Table 5 shows the results for each classifier when implemented in WEKA. The results of Random Forests differ slightly from the results in Malekipirbazari and Aksakalli (2015) as hyperparameter tuning used too much storage in WEKA and was not possible. The Support Vector Classifier within WEKA ran for several hours and failed to complete thus Sequential Minimal Optimization (SMO) (Platt, 1998) which is essentially a fast SVM training algorithm is used instead. Overall, Random Forests perform the best across all metrics aside from the AUC score, and SVM performs the worst. Logistic Regression performs best when measured using the AUC metric, however, its performance is sub-optimal due to a high correlation between explanatory variables. As shown in Figure 9 in the Appendix, quite a few variables are negatively correlated with each other. This reduces the performance of Logistic Regression and should be considered in future research. When comparing results using the F1 metric, the results are similar to those when compared using the Accuracy. That is, Random Forests perform the best followed by KNN.

Table 3: Results from WEKA

	Accuracy (%)	AUC	RMSE	F1	TP Rate Good	TP Rate Bad	FP Rate Good	FP Rate Bad
KNN	70.1	0.54	0.55	0.70	0.82	0.26	0.74	0.18
Logistic Regression	54.5	**0.69**	0.51	0.59	0.49	0.77	0.23	0.51
SVM	53.3	0.63	0.68	0.57	0.47	**0.78**	**0.22**	0.53
Random Forest	**74.5**	0.67	**0.42**	**0.74**	**0.85**	0.34	0.66	**0.15**
XGBoost	55.8	**0.69**	0.50	0.60	0.51	0.76	0.25	0.49

Note: the numbers in bold represent the best technique for each valuation metric.

XGBoost performs poorly, producing significantly worse results than Random Forests. It performs similarly to Logistic Regression and worse than KNN when classifying *Good* borrowers. This could be because the "mlr" classifier within R may not work well in conjunction with the cost-sensitive classifier package. This is one of the main reasons why the results are replicated in Python using different techniques. Additionally, the parameters tuned namely, learning rate, max depth, and the number of trees helped in reducing overfitting however, this also reduced the performance of the classifier. Table 4 shows the results of XGBoost with 1000 trees used and a max depth of 10. The results are notably better, however, underperform when considering the TP Rate of *Bad* borrowers. Moreover, there is significant overfitting as the results on the training set are notably higher than the test set. Thus, tuning the parameters of XGBoost reduces overfitting however comes at the cost of reducing performance.

Table 4: Training and Test Results XGBoost

	Accuracy (%)	AUC	RMSE	F1	TP Rate		FP Rate	
					Good	Bad	Good	Bad
Training Set	87.9	0.98	0.29	0.89	0.86	0.97	0.03	0.14
Test Set	69.6	0.66	0.46	0.71	0.76	0.46	0.54	0.24

5.2 Results from Python

Using a stratified train test split with a test size of 20% the results of XGBoost and Random Forests are replicated. This is method is used over cross-validation as it maintains the ratio of positive and negative instance between the train and test sets. Thus by using this method, we compare whether it has a positive effect on the results. Random oversampling is used to correct the issue of imbalance. As before the variables *FICO* and *LCGrade* are removed for fairness. The results are replicated 5 times using different random states each time and the average across results is taken. This is done to ensure these results are comparable to those in WEKA where 5-fold cross-validation was used. Furthermore, the learning rate is set to 0.1, the max depth is set to 5, and the number of trees is set to 50. For Random Forests, max depth is set to 25, max features set to 5, and the number of trees grown is 80 as done in the paper by Malekipirbazari and Aksakalli (2015).

Table 5: Python Results

	Accuracy	AUC	RMSE	F1	TP Rate	FP Rate
Random Forests	**67.2**	0.55	**0.47**	**0.87**	**0.95**	0.86
XGBoost	65.0	**0.64**	0.59	0.75	0.66	**0.39**

The results are quite different from those found in WEKA, as XGBoost performs better when

using the AUC metric and the FP Rate. Random Forests has a very high true positive rate, however, the false positive rate is also significantly higher than that of XGBoost. The use of a stratified train test split did not have a notable positive effect on the results. Thus, with these mixed results it is difficult to identify a clearly better model. Combining these results with those found in WEKA, it is clear that XGBoost does not outperform Random Forests for this dataset.

5.3 Comparison against previous methods

In this subsection, we compare the results found from the proposed models against existing methods. Specifically, we recreate the results from Emekter, Tu, Jirasakuldech and Lu (2015) who used the same data from Lending Club however used only four features to create a binary logistic model. Namely, these variables are the *FICO* score, *LCGrade*, *DTI*, and *RevolvingUtilizationRate*. We compare these results with the all feature Random Forest model excluding the variables *FICO* score and *LCGrade*. Moreover, we test whether 4-Feature Random Forests or XGBoost beats the proposed Logistic model. All 4 feature models were trained with a 5:1 cost ratio for fairness and the parameters of Random Forests are set to their default values. We see the all feature and 4-Feature Random Forests models outperform Logistic Regression by a large margin. Specifically, we note that 4-Feature Random Forests is much better at classifying Good borrowers with a TP rate of 0.79 compared to 0.45 of Logistic Regression. 4-Feature XGBoost performs similarly to Logistic Regression and is again significantly worse than Random Forests. Having said this, XGBoost performs best in the TP Rate of *Bad* borrowers as well as having the lowest misclassification rate of 0.22 for *Good* borrowers.

Table 6: Comparison of 4-Feature models

	Accuracy (%)	AUC	RMSE	F1	TP Rate		FP Rate	
					Good	Bad	Good	Bad
All Feature Random Forest	**75.8**	**0.68**	**0.42**	**0.75**	**0.88**	0.30	0.70	**0.12**
4-Feature Random Forest	69.9	0.61	0.45	0.70	0.80	0.32	0.68	0.20
4-Feature XGBoost	48.4	0.64	0.50	0.52	0.41	**0.78**	**0.22**	0.59
4-Feature Logistic Regression	50.5	0.66	0.52	0.54	0.44	0.77	0.23	0.56

Note: the numbers in bold represent the best technique for each valuation metric.

6 Conclusion

Cost-sensitive classification is a widely used method in fields such as medicine and finance. This paper replicated and extended the work by Malekipirbazari and Aksakalli (2015) which explored several popular classifiers in the context of classifying good borrowers. As an extension, an emphasis

was placed on the comparison between bagging and boosting methods namely, comparing the performance of Random Forests and XGBoost which came out after the publication of Malekipirbazari and Aksakalli (2015)'s paper. The dataset used was taken from the Social Lending platform and a cleaned version of this was used. This dataset contained 18 features including the dependant variables - the status of the loan. Of these 18, 3 were categorical and the remainder were numerical. 80% of borrowers repaid their loan and the remaining defaulted in some way resulting in a notably imbalanced dataset. To account for this a cost-sensitive method was used imposing a 5:1 cost ratio for classifying bad borrowers as good. All classifiers were implemented using the WEKA software and k=5 folds cross-validation was used. To evaluate the performance of the classifiers, the Accuracy, ROC-AUC, F1 Score, RMSE, and Recall were used. The results showed very similar results to those in Malekipirbazari and Aksakalli (2015) with minor differences possibly due to different random seeds being used.

XGBoost performed worse than Random Forest and took longer to run. This was thought to be due to the cost sensitivity not working well in conjunction with XGBoost. The results were replicated in Python, this time using a random oversampling method to balance the classes however, the results were not significantly better. Due to time constraints, the max depth, learning rate, and the number of trees were tuned while leaving the remaining parameters at their default value. The tuned parameters helped in reducing the overfitting of the model however also hindered the performance somewhat. The answer to the research question posed is that XGBoost does not outperform Random Forest when assessing risk in P2P lending.

Due to time constraints and computational efficiency, a few limitations are present within this paper. Hyperparameter optimization has a large effect on the performance of classifiers, however, this was only possible to do in Random Forests and XGBoost. Not all hyperparameters were tuned for these classifiers and none were tuned for Logistic Regression, KNN, or SVM. Furthermore, as Logistic Regression does not perform well where there is multicollinearity, several variables could be removed to reduce this. Additionally, for KNN, using a value of K=1 results in overfitting, and thus optimizing this value could improve performance. For future research, Random Forest and XGBoost could be compared over other different datasets and over different platforms such as Prosper, another popular P2P lending site. Moreover, alternative boosting methods can be compared with XGBoost such as ADABoost, another Gradient Boosting method.

References

Bachmann, A., Becker, A., Buerckner, D., Hilker, M., Kock, F., Lehmann, M., ... Funk, B. (2011). Online peer-to-peer lending-a literature review. *Journal of Internet Banking and Commerce*, *16*(2), 1.

Breiman, L. (1996). Bagging predictors. *Machine learning*, *24*(2), 123–140.

Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5–32.

Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, *2*(2), 121–167.

Chen, T. & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785–794).

Cortes, C. & Vapnik, V. (1995). Support vector machine. *Machine learning*, *20*(3), 273–297.

Cover, T. & Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, *13*(1), 21–27.

Emekter, R., Tu, Y., Jirasakuldech, B. & Lu, M. (2015). Evaluating credit risk and loan performance in online peer-to-peer (p2p) lending. *Applied Economics*, *47*(1), 54–70.

Fan, J., Wang, X., Wu, L., Zhou, H., Zhang, F., Yu, X., ... Xiang, Y. (2018). Comparison of support vector machine and extreme gradient boosting for predicting daily global solar radiation using temperature and precipitation in humid subtropical climates: A case study in china. *Energy conversion and management*, *164*, 102–111.

Freund, Y., Schapire, R. E. et al. (1996). Experiments with a new boosting algorithm. In *icml* (Vol. 96, pp. 148–156).

Kim, M.-J., Min, S.-H. & Han, I. (2006). An evolutionary approach to the combination of multiple classifiers to predict a stock price index. *Expert Systems with Applications*, *31*(2), 241–247.

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

Maclin, R. & Opitz, D. (1997). An empirical evaluation of bagging and boosting. *AAAI/IAAI*, *1997*, 546–551.

Malekipirbazari, M. & Aksakalli, V. (2015). Risk assessment in social lending via random forests. *Expert Systems with Applications*, *42*(10), 4621–4631.

Nielsen, D. (2016). *Tree boosting with xgboost-why does xgboost win" every" machine learning competition?* (Unpublished master's thesis). NTNU.

Platt, J. (1998). *Fast training of support vector machines using sequential minimal optimization, in,*

b. scholkopf, c. burges, a. smola,(eds.): Advances in kernel methods-support vector learning. MIT Press.

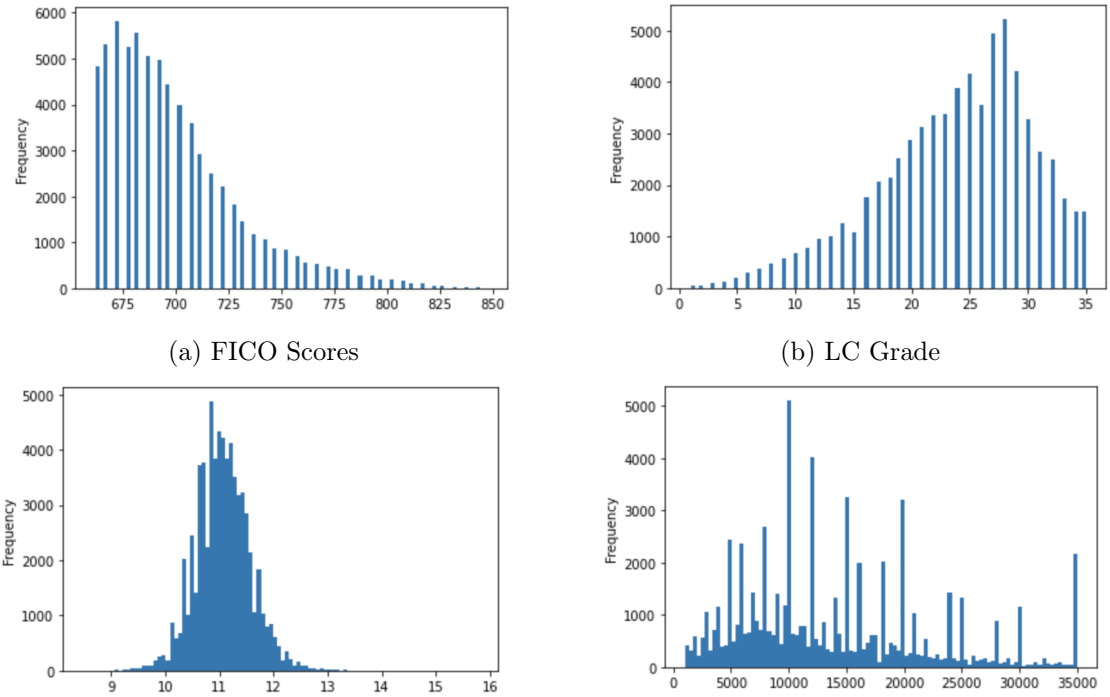
Witten, I. H. & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques* (2nd ed.). San Francisco: Morgan Kaufmann.

A Data Exploration

Table 7: Feature Description

Feature	Description
Annual Income	Annual income of borrower *
Credit Age	Date of first credit line opened converted to months*
Delinquencies	Number of deliquencies in the last two years
Employment Length	Employment length in years taking values between 1 and 10
Home Ownership	categorical variables with values: Rent, Own, and Mortgage
Inquiries	number of credit inquiries in the last six months
Loan Amount	size of loan in dollars where maximum is $35000
Loan Purpose	categorical variable with possible values: ebt Consolidation, Home Improvement, Credit Card, Moving, Small Business, Car, Major Purchase, Vacation, Medical, Renewable Energy, House, Wed- ding, and Other.
Open Accounts	number of open credit lines on the borrower's credit file.
Total Accounts	total number of credit lines on borrowers file.
Term	Number of monthly payments of loan taking values between 36 and 60
Features created by taking ratio of other features	
Debt to Income Ratio (DTI	Ratio of the borrower's total monthly debt payments to the borrower's monthly income
Income to Payment Ratio	Ratio of the loan's monthly pay- ments to monthly income*
Revolving Utilization Rate	he amount of credit the borrower is using relative to all available revolving credit
Revolving Income Ratio	Ratio of revolving credit balance to the borrower's monthly income
Additional attributes	
FICO Score	The standard credit scores that are used in majority of the lending decisions in the US
LC Grade	grade created by LC ranging from A to G using a proprietary algorithm based on loan characteristics and risk assessment of borrower. Each grade is split into 5 levels such as A1-A5. These were converted to numerical values such that 1 = A1 and 35 = G5.

$*$ =The natural log used in calculation



(a) FICO Scores

(b) LC Grade

(c) Annual Income

(d) Loan Amount

Figure 8: Histograms for FICO Scores, LC Grade, Annual Income, and Loan Amount

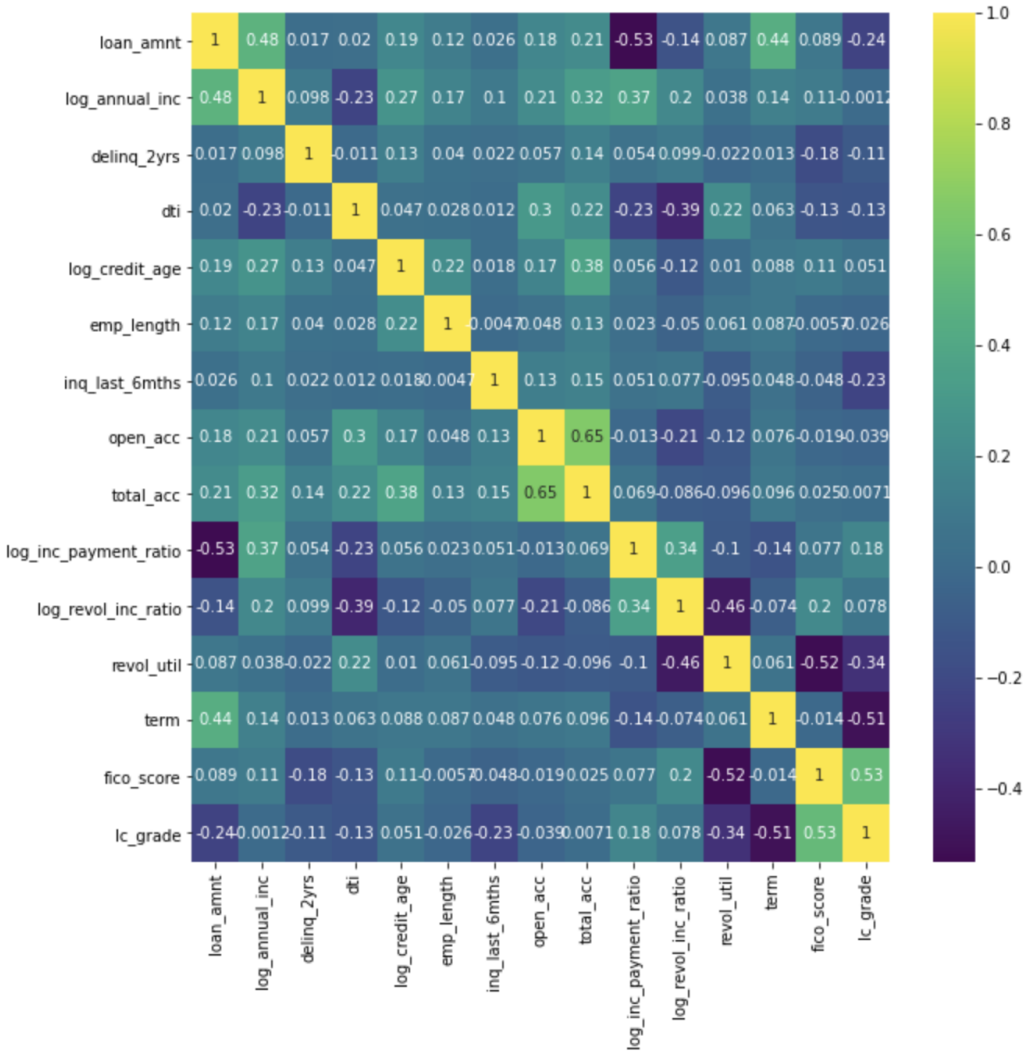


Figure 9: Correlation Heatmap of all Variables

B Algorithms

Algorithm 1 Random Forest Algorithm

Input: D = Training Data

Output: Bagged Class label for input data

To generate c classifiers: **for** $i = 1$ to c **do**

Randomly sample the training data D with replacement to produce D_i

Create a root node, N_i containing D_i

Call BuildTree(N_i)

end

BuildTree(N):

if N contains instances of only one class **then return**;

else

Randomly select x% of the possible splitting features in N

Select the feature F with the highest information gain to split on

Create f child nodes of N, $N_1, ..., N_f$, where F has f possible values $(F_1, ..., F_f)$;

for $i = 1$ to f **do**

Set the contents of N_i to D_i, where D_i is all instances in N that match F_i

Call BuildTree(N_i)

end

Algorithm 2 Gradient Boosting Algorithm

Input: training set $\{(\mathrm{x}_i, y_i)\}^n$, a differentiable loss function $L(y, F(x))$, number of iterationsM

Output: $\mathrm{F}_M(x)$

Initialize model with a constant value: $\mathrm{F}_0(x) = arg\,min_\gamma \sum_{i=1}^{n} L(y_i, \gamma)$

for $m = 1$ to M **do**

Compute $pseudo - residuals$:

$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1(x)}} \quad for \quad i = 1...n$

Fit a base learner (eg tree) $h_m(x)$ to pseudo-residuals, ie train model using training set $\{(x_i, r_{im})\}^n$

Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$\gamma_m = arg\,min_\gamma \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$

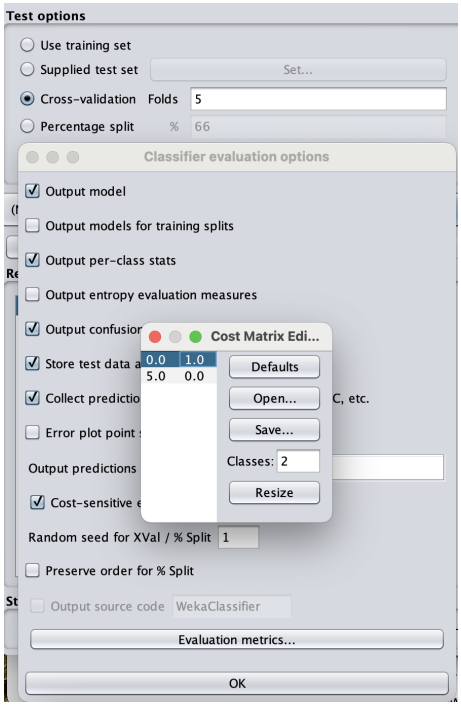
end

Update the model: $\mathrm{F}_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$.

C WEKA implementation

C.1 All-feature implementation

1. Load the cleaned data "*LCClean_Numeric*"as a csv file into WEKA.
2. Remove the variables *fico_score* and *lc_grade*
3. Use the "ReplaceMissingValues" filter under the attribute of unsupervised filters and apply to the variable *revol_util*.
4. For all classifiers except Random Forests apply the "NominalToBinary" filter under filters -> unsupervised -> attribute to the variables *purpose* and *home_ownership*. This is not required for Random Forests as it can handle categorical data. Note: For XGBoost, the RPlugin is required to be installed on the device.
5. Under test options in the Classify tab, set Cross-validation Folds = 5.
6. In More options under Test options tick the cost-sensitive evaluation box and set classes to 2 and bottom left box = 5 and click Resize as shown below: This uses costs sensitivity for the



test set where misclassification of a bad borrower is 5 times worse than a misclassification of a good borrower.

7. Under the result list in Classify load each model and run for the dataset imported, which has been preprocessed in steps 2-4.

C.2 4-feature implementation

1. Remove all variables from the same dataset ("$LCClean\_Numeric$") except for dti, $revol\_util$, $fico\_score$, lc_grade, $loan\_status$

2. Choose the filter "ReplaceMissingValues" under attribute of unsupervised methods of the filters. Apply this to $revol\_util$.

3. Repeat steps 5-7 from the all-feature implementation, loading the 4-feature models only.