Classification Methods for Predictive Credit Scoring

Deeksha Doddahonnaiah, Sahil Gandhi, and **Donald Hamnett**
Northeastern University
College of Computer and information Science
440 Huntington Ave #202, Boston, MA 02115

Abstract

The aim of this project was to implement an effective classification method for credit-scoring. We used a real company's data-set, and each author focused on a different classifier. The classifiers used were Support Vector Machines (SVMs), Random Forests, and ArtificialNeural Networks (ANNs). Each of these algorithms presented issues which we attempted to solve with a variety of techniques. A particular issue over each classification implementation was imputing "missing" data points, and overcoming the imbalance of classes in the dataset. Experimenting with different solutions lead to meaningful insights, with varying success in the classification task.

Introduction

An issue that often arises for companies in the financial service industry is qualifying potential borrowers. Companies receive leads and applications via several channels, whether it be through marketing campaigns, cold-calling, or direct customer inquiries. Any amount of time and resources spent in pursuing a unfruitful account is lost time and profit pursuing a better qualified loan candidate. This becomes a serious issue for the company providing the service, because a significant amount of time and resources go into processing each lead. When application processing is a bottleneck in the company's pipeline, inefficiency in qualifying these customers has an adverse effect on revenues, due to both direct expenditures and opportunity cost. We see customer prioritization as a two-fold issue in relation to the customer relationship timeline: credit scoring and predictive lead scoring. The company will first decide which leads to act on initially, then qualify these leads by obtaining credit and other financial data. Credit scoring is an area where there has been considerable research and success[1], due to the convenience of having hard financial data off of which to make predictions. It is our aim to replicate these successes by offering a classification solution to a lender who is experiencing this very issue.

By using a collection of classifiers and ensemble methods, we were able to offer the beginnings of an effective classification pipeline which will only improve with time as the company takes in more data. We used three of the most prevalent classifiers used in industry today: Support Vector Machines, Random Forests, and Artificial Neural Networks, and tailored them to the task at hand, while remaining general enough to be extensible to unknown data. We believe that our work will help our client maximize their earning potential and match more qualified and deserving borrowers with a responsible lender.

Copyright © 2017, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Related Work

Predictive lead scoring is not among the well known problems researchers have tried to solve. However there has been significant studies done on similar problems, credit risk analysis, customer churn prediction, loan default prediction, customer choice modeling, to name a few. The work done in the above areas are along the similar lines to our project, and we have taken advantage of their studies as we noticed that our data overlaps with the data used for these studies.

The closest study to ours is the work done by Milad Malekipirbazari and Vural Aksakalli [2] have employed Random Forests and SVM to identify true creditworthiness of a potential loan borrower. They have used random forest feature importance to show that financial features such as past financial history, existence of delinquent accounts, debt to income ratio (DTI) are significant to compute risk score. [3] demonstrates the use of feature importance to prove that past customer behaviors, and financial information play an important role in predicting future customer behavior and profitability.

ANNs have well-documented used in credit scoring applications [4, 5], and this success drove the decision to choose them as one of our classifiers. in addition, though details are typically not publicly available, several proprietary solutions have reported positive results in applying ANNs to predictive-lead scoring[6], and we hoped that this would translate to credit-scoring for our dataset, as lead-scoring similarly must make insights based on very limited data. One work, though not related to credit-scoring, which greatly influenced my approach was "Ensembling neural networks: Many could be better than all"[7], because it gave the intuition to apply ensemble methods to neural networks, which ended up being the final approach used. Lastly, one very interesting application of ANNs to credit-scoring was "Predicting mortgage default using convolutional neural networks"[8]. Although our dataset was too small, and experi-

ence too minimal, to apply these techniques, it is an exciting look into the future of research in credit-scoring and novel use cases for a variety of neural network models.

Background

Support Vector Machines

in this section we will briefly discuss the basic SVM concepts for typical two-class classification problems. Given a training set of instance-label pawers $(x_i, y_i), i = 1, 2, \ldots, m$ where $x_i \in \mathbb{R}^n$ and $y_i \in \{+1, -1\}$, SVM finds an optimal separating hyperplane with maximum margin by solving the following optimization problem:

$$\min_{w,b} \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

$$\text{subject to: } y_i(\langle \mathbf{w}.\mathbf{x_i}\rangle + b) - 1 \geq 0$$

To solve this problem, we must find the saddle point of the Lagrange function:

$$L_p(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{m}(\alpha_i y_i(\langle \mathbf{w}.\mathbf{x_i}\rangle + b) - 1)$$

where α_i represent the Lagrange multipliers, hence $\alpha_i \geq 0$. Since L_p must be minimized with respect to the prwemal variables $\mathbf{w}$ and b, and maximized with respect to the non-negative dual variable α_i, the minimum saddle-point must be found. L_p is transformed to the dual Lagrangian $L_D(\alpha)$ by differentiating with respect to $\mathbf{w}$ and b, and introducing the Karush Kunh-Tucker (KKT) condition for the optimum constrained function [9].

$$\max_{\alpha} L_D(\alpha) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y_i y_j \langle \mathbf{x_i}, \mathbf{x_j}\rangle$$

$$\text{subject to: } \alpha_i \geq 0 : \forall i = 1, 2, \ldots, m \text{ and } \sum_{i=1}^{m}\alpha_i y_i = 0$$

The optimal hyperplane can be found by maximizing the dual Lagrangian with respect to the Lagrangian multipliers α_i. Hence, the optimal hyperplane decision function $f(x) = sgn(\langle \mathbf{w}^*.\mathbf{x_i}\rangle + b^*)$ can be written as

$$f(x) = sgn(\sum_{i=1}^{m} y_i\alpha_i^*\langle \mathbf{x_i}.\mathbf{x}\rangle + b^*)$$

Typically, only a small subset of Lagrange multipliers is greater than zero. These vectors are called the support vectors, as the optimal decision hyperplane depends on them. Geometrically, these support vectors are the closest to the optimal hyperplane. All above concepts deal with classifying linearly separable data. However, they can also be extended to the non-linearly separable case. The concept of slack variables is introduced and the problem changes to finding a hyperplane that provides the minimum number of training errors [10]. The following equation formalizes the concept:

$$\min_{w,b,\xi} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{m}\xi_i$$

$$\text{subject to: } y_i(\langle \mathbf{w}, \mathbf{x_i} + b) + \xi_i - 1 \geq 0 \text{ and } \xi_i \geq 0$$

where ξ_i is the non-negative slack variable and C is the penalty parameter on the training error. The optimization problem can be solved using the Lagrangian method which is almost equivalent to the method for solving the optimization problem of linearly separable case.

$$\max_{\alpha} L_D(\alpha) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{we,j=1}^{m}\alpha_i\alpha_j y_i y_j \langle \mathbf{x_i}, \mathbf{x_j}\rangle$$

$$\text{subject to: } 0 \leq \alpha_i \leq C : \forall i = 1, 2, \ldots, m \text{ and } \sum_{i=1}^{m}\alpha_i y_i = 0$$

This nonlinear version of SVM maps the originalfeature space into a higher-dimensional feature-space via a mapping function Φ and in the dual Lagrange, the inner products are replaced by a Kernel function as shown below:

$$\langle \Phi(\mathbf{x_i}).\Phi(\mathbf{x_j})\rangle := k(\mathbf{x_i}, \mathbf{x_j})$$

The nonlinear SVM dual Lagrangian is similar to that in the linear generalized case:

$$L_D(\alpha) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{we,j=1}^{m}\alpha_i\alpha_j y_i y_j k(\mathbf{x_i}, \mathbf{x_j})$$

Following the steps performed in the linear generalized case, we obtain the decision function:

$$f(x) = sgn(\sum_{i=1}^{m} y_i\alpha_i^* k(\langle \mathbf{x_i}.\mathbf{x_j}\rangle) + b*)$$

The most commonly used Kernel functions are [11]:

1. Linear kernel function: $K(\mathbf{x_i}, \mathbf{x_j}) = \langle \mathbf{x_i}.\mathbf{x_j}\rangle$
2. Polynomial kernel function: $K(\mathbf{x_i}, \mathbf{x_j}) = (\langle \mathbf{x_i}.\mathbf{x_j}\rangle^d + 1)$
3. Radial Basis Function kernel function: $K(\mathbf{x_i}, \mathbf{x_j}) = \exp(-||\mathbf{x_i} - \mathbf{x_j}||^2\frac{1}{\sigma^2})$
4. Sigmoid kernel function: $K(\mathbf{x_i}, \mathbf{x_j}) = \tanh[k\langle \mathbf{x_i}.\mathbf{x_j}\rangle + \nu]$, where $k > 0$ and $\nu < 0$

Vapnik introduced the kernel space theory for the linearly non-separable problem [2]. The above SVMs are designed for binary classification. The SVM for multiclass classification is by constructing and combining several binary classifiers. The former approach mainly includes three methods: One-Against-All (OAA) [12], One-Against-One (OAO) [13] and Binary-Tree-Architecture (BTA) [14].

Random Forests

Decision trees are known to be one of the popular classification technique, where it predicts class probabilities at the end of the tree branches. Decision trees start with the root node and gradually build branches, making splits based on certain features at each node, till it reaches the terminal node called leaves. The root node starts with all samples and the splits are binary. In random forests the splitting is done based on Gini index, the split is done on the attribute with highest Gini index.

The Gini Index at an internal tree node is calculated as follows; for a candidate (nominal) split attribute X_i, denote

possible levels as $L_1, \ldots, L_J$ Gini Index for this attribute is calculated as:

$$G(X_i) := \sum_{j=1}^{J} Pr(X_i = L_j)(1 - Pr(X_i = L_j))$$
$$= 1 - \sum_{j=1}^{J} Pr(X_i = L_j)^2$$

In this project we use a popular decision tree proposed by [15], called random forests. Random forests work by creating multiple decision trees and each tree builds on a random set of features. Each tree is grown on a bootstrap sample of training set. A subset of k predictors is used to split the samples in each of the decision trees. Typically, k is much smaller than the actual number of features. The predictions are made by calculating the highest votes from all the trees.

The random forest algorithm is as follows:

1: **procedure** RANDOM FOREST
2: **repeat**
3: **repeat**
4: Randomly select k features from total m features where $k << m$
5: Among the k features, calculate the node d using the best split point
6: Split the node into daughter nodes using the best split
7: **until** i number of nodes has been reached
8: **until** Forest has been built to n trees
9: Take the test features and use the rules of each randomly created decision tree to predict the outcome and stores the predicted outcome (target)
10: Calculate the votes for each predicted target
11: Consider the high voted predicted target as the final prediction from the random forest algorithm
12: **return** Final Prediction

Random forest have many advantages compared to decision trees. The most powerful aspect about random forests is variable importance ranking which estimates the predictive value of variables by scrambling the variable and seeing how much the model performance drops. Random forests also reduce overfitting, as it averages multiple trees. It is also faster than boosting and bagging.

ArtificialNeural Networks

Artificial Neural Networks have their origins in 1943, with the McCollough-Pitts Threshold Logic Unit (TLU), a mathematical interpretation how formal logic is carried out in the human brain[16]. The TLU representation of a neuron, is as a unit that takes in a set of inputs and calculates a weighted sum. This weighted sum would then be processed as:

$$f(\mathbf{x}) = \begin{cases} 0, & \left(\sum_i w_i x_i\right) + b < \text{threshold} \\ 1, & \text{Otherwise} \end{cases}$$

Figure 1: TLU Activation

Any discrete binary task could be represented by these units, but their usefulness was simply as a model of the brain and offered no deductive improvements upon formal logic.

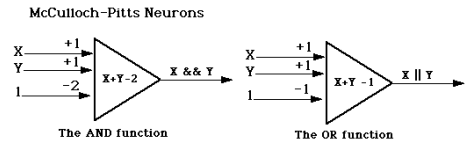

Figure 2: The McCollough-Pitts Neuron[17]

in 1958, the precursor of the modern-day neural network was born with Rosenblatt's development of the "perceptron." Expanding on the MuCollough-Pitts representation of a neuron, his perceptron was a unit that similarly took in a set of inputs and calculated a weighted sum, activating with the sign function.

$$sgn(\mathbf{x}) = \begin{cases} -1, & \left(\sum_i w_i x_i\right) + b < 0 \\ +1, & \text{Otherwise} \end{cases}$$

Figure 3: The Sign Function

The key insight of Rosenblatt was that opposed to weights being fixed, they were adjustable parameters with positive and negative weights, and could hence be learned, hoiver, his perceptron algorithm[18] was eventually criticized for being capable of charactering solely linearly-separable decision boundaries and was most infamously incapable of computing the exclusive-or function.

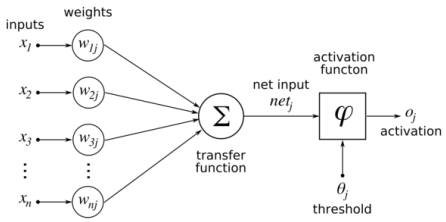

Figure 4: The Rosenblatt Perceptron[**mitchell˙1997**]

The full history of perceptrons is outside the scope of this report, but after a period of falling out of fashion, the perceptron has made a massive resurgence in recent years due to the development of the ANN, where several perceptrons are combined in parallel and/or in layers to compute complex predictive models that can be applied to non-linearly separable decision boundaries. While the ANN was first developed in the 1960s, notably ADALINE and MADELINE[19], it was not until the recent increases in both computational power and available data that their widespread use was adopted. Since then, these networks have had success in a vast collection of problems, ranging from natural language processing with recurrent neural networks, to computer vision with convolutional neural networks (CNNs). However, this classification problem focuses on perhaps the simplest use case of ANNs, the simple fully-connected feed-forward ANN.

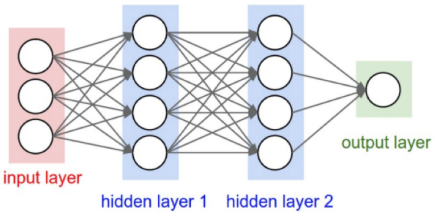

Figure 5: A Fully Connected Feed-Forward ANN[20]

Project Description

In order to find the best model to fit our data, we combined the simpler algorithms mentioned in the previous section with various optimization techniques. The techniques we use are, Grid Search with k-fold Cross Validation for parameter estimation.

Grid Search + k-fold Cross Validation

The traditional way of performing hyper-parameter optimization has been grid search, or a parameter sweep, which is simply an exhaustive searching through a manually specified subset of the hyper-parameter space of a learning algorithm. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set or evaluation on a held-out validation set. The advantages of cross validation are that the impact of data dependency is minimized, and the reliability of the results can be improved [21]. In grid search, arbitrary values of parameters are chosen and tried, and the one with the best cross-validation evaluation metric is chosen. The procedure is as follows:

1: **procedure** GRID SEARCH K-FOLD
2: Consider a grid space of parameters
3: **for** each permutation of hyper-parameter values in the search space **do**
4: conduct k-fold cross validation on the training set
5: Choose the parameter that leads to the best evaluation criteria
6: **return** The best parameter to create a model of the predictor

In addition to these techniques, we also used a few different flavors of the simple algorithms as described below:

Support Vector Machine + Grid Search (SVM+GS)

Proper parameter settings can improve the accuracy of the SVM classifier. Depending on the kernel function, there are various parameters to be determined in the SVM model. For example, in RBF kernel, C and gamma (γ) need to be determined. The grid search approach is best method of determining the parameters for a given kernel [22]. However, in order to guarantee that the results are correct and can be generalized to make predictions on unseen data, the dataset is further partitioned into multiple training and validation sets via k-fold cross validation. The approach mentioned in the previous section is followed and the parameters are tuned.

Support Vector Machine + Grid Search + F-score (FS+SVM+GS)

In addition to parameter setting, feature selection can also improve accuracy of the SVM classifier. F-score is a simple technique which measures the discrimination of two sets of real numbers [23]. Given training vectors $\mathbf{x_k} : k \in [1, 2, \ldots, m]$ if the number of positive and negative instances are $n+$ and $n-$, respectively, then the F-score of the i^{th} feature is defined as [23]:

$$F(i) \equiv \frac{\left(\bar{x}_i^{(+)} - \bar{x}_i\right)^2 + \left(\bar{x}_i^{(-)} - \bar{x}_i\right)^2}{\frac{1}{n_+ - 1}\sum_{k=1}^{n_+}\left(x_{k,i}^{(+)} - \bar{x}_i^{(+)}\right)^2 + \frac{1}{n_- - 1}\sum_{k=1}^{n_-}\left(x_{k,i}^{(-)} - \bar{x}_i^{(-)}\right)^2},$$

Figure 6: F-Score

where $\overline{x}_i$, $\overline{x}_i^{(+)}$, $\overline{x}_i^{(-)}$ are the average of the i^{th} feature of the whole, positive, and negative data sets, respectively; $\overline{x}_{k,i}^{(+)}$ is the i^{th} feature of the k^{th} positive instance, and $\overline{x}_{k,i}^{(-)}$ is the i^{th} feature of the k^{th} negative instance. The numerator indicates the discrimination between the positive and negative sets, and the denominator indicates the one within each of the two sets. Larger F-score indicates that the feature is more likely to be discriminative. The procedure is as follows [23]:

1: **procedure**
2: Compute F-score for every feature
3: Pick arbitrary thresholds
4: Cut thresholds into low and high F-scores
5: **for** Each threshold **do**
6: **repeat**5 times
7: Drop features with F-score outside the threshold
8: Randomly split the training data into Xtrain and Xvalid
9: Let Xtrain be the new training data. Use a modeled SVM procedure with chosen parameters to obtain a predictor; use the predictor to predict Xvalid
10: **until done**
11: Calculate the average validation error
12: Choose the threshold with the lowest validation error
13: Drop features with F-scores that don't fit the selected threshold.
14: Apply the SVM procedure on the new feature space
return

Ensemble SVM (SVM+ENS)

An ensemble of classifiers is a collection of several classifiers whose individual decisions are combined in some way to classify the test examples [10, 11]. It is known that an ensemble often shows much better performance than the individual classifiers that make it up. Hansen et. al. [24] talks about why the ensemble shows better performance than individual classifiers. SVM is known to show good generalization performance, but as will be evident in future sections, the simple SVM model does not work well with the chosen dataset. To overcome this limitation, we propose to use an ensemble of SVMs, among other algorithms. We chose the Bagging classifier technique to construct the SVM ensemble. In bagging, several SVMs are trained independently

via a bootstrap method and then they are aggregated using a combination technique. We generate k training datasets, by randomly re-sampling, with replacement, from the training data. Each replicate training set is then used to train a certain SVM.

Random forests + GridSearch (RF+GS)

Random forests have various parameters one can tune in order to prevent overfitting; forest size, tree depth and feature subset selection being the most important ones. As explained earlier, GridSearch is a way to define a grid of hyperparameters to figure out the best parameters for the model. With respect to Random Forests, we have the following parameters to tune:

- n estimators = number of trees in the forest
- max features = max number of features considered for splitting a node
- max depth = max number of levels in each decision tree
- min samples split = min number of data points placed in a node before the node is split
- min samples leaf = min number of data points allowed in a leaf node

Weighted Random Forests: (WRF + GS)

One approach to handle class imbalance data is to use the idea of cost-sensitive analysis. Random forest classifiers are more biased towards majority class, and hence we place heavier weights on minority class as proposed by Chao Chen, Andy Liaw, Leo Breiman [25] . This results in penalizing the misclassification of minority class more heavily. These class weights are applied to the node splitting criterion and the aggregation procedure in order to ensure that the minority class is weighted equally. The class prediction of each terminal node is determined by "weighted majority vote"; i.e., the weighted vote of a class is the weight for that class times the number of cases for that class at the terminal node. . The final class prediction for RF is then determined by aggregating the weighted vote from each individual tree, where the weights are average weights in the terminal nodes. We use the weighted random forests with the grid search parameters chosen in RF+GS.

Balanced Random Forests: (BRF + GS)

Balanced random forests make the class priors equal by oversampling the minority class in learning highly imbalanced data. The Balanced Random Forest (BRF) algorithm as proposed by Chao Chen, Andy Liaw, Leo Breiman [25] is shown below: For each iteration in random forest, we bootstrap sample from the minority class. Randomly draw the same number of cases, with replacement, from the majority class. Induce a classification tree from the data to maximum size, without pruning. The tree is induced with the CART algorithm, with the following modification: At each node, instead of searching through all variables for the optimal split, only search through a set of mtry randomly selected variables. Repeat the two steps above for the number of times desired. Aggregate the predictions of the ensemble and make the final prediction.

However, changing the balance of negative and positive training samples has little effect on the classifiers produced by decision-tree learning methods (Elkan, 2001)[25]. In summary, according to Chen et al.'s study (2004)[25], there is no clear winner between weighted random forests and balanced random forests.

Ensemble Neural Networks

Using the Keras deep learning library in Python with its functional API made it incredibly easy to build an ANN. Below is the pseudocode used to build a fully-connected feed-forward network.

procedure GETNETWORK(NumberOfHiddenLayers)
    inputLayer ← $input$(NumberOfFeatures)
    x ← inputLayer
    for $i = 1$ to NumberOfHiddenLayers **do**
        x ← $HiddenLayer$(NumberOfFeatures)(x)
    OutputLayer ← $Layer(1)$(x)
    ANN← Model(inputLayer, OutputLayer)
    Compile(ANN)
return ANN

in all networks, we used the rectified linear unit (ReLU) for the hidden layer activations, and the logistic sigmoid with a threshold of 0.5 for the output activation.

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

$$ReLU(x) = \max(0, x) \tag{2}$$

Figure 7: The sigmoid and ReLU functions

Using the default parameters for the Adam [26] optimizer, we build ansemble networks based on this architecture.

Data

For this project, we used a dataset that includes anonymized information of business loan applications. The dataset includes all the information that a normal loan application would need; including, but not limited to the company and individual demographic data, corporate bank transactions, credit information data, and detailed loan-lifecycle data. The reason for choosing this dataset was that since it contained data mined from a live enterprise venture, it emulates the real life application of predictive lead scoring. However, the dataset was not without issues; It consisted of a large amount of missing values and the classes labels were skewed. One class label was solely responsible for 98% of the the data. Most of the classifiers failed to predict anything using this data.

In order to fix the missing values, we imputed relevant information based on some preliminary data analysis. We take advantage of domain knowledge to correlate the features and their relevance to the target variable. The heat-map below show the correlation among variables. For features where we couldn't apply domain knowledge, a simple arithmetic mean of was imputed in order to fill the values.

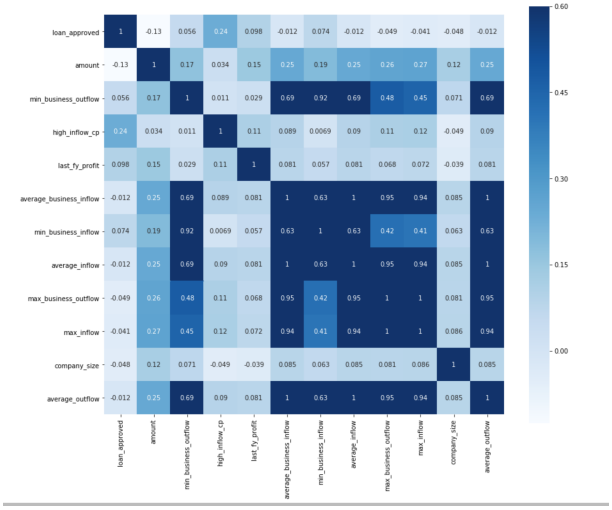

Figure 8: HeatMap

SMOTE: Synthetic Minority Over-sampling TEchnique

The problem of skewed classes was looked at in 2 ways, one was to make the classifier work around the skewness by setting appropriate weights for class labels and allowing and the other was to generate synthetic data to balance the classes. We use SMOTE to oversample the data. Synthetic Minority Over-sampling TEchnique, or simply SMOTE, over-samples the minority class by creating synthetic minority class data. The algorithm works with a set of samples s, with f features in the feature space of the data. The algorithm then takes a sample from the dataset and considers its k nearest neighbors. To create synthetic data, we consider the vector between one of the those k neighbours , and the current data point. We then multiply this vector by a random number between 0 and 1 to obtain the new data point. For our dataset, we have created roughly 8% more data, above the existing 2% for minority class, which puts 90% of data in majority class, and 10% for minority class.

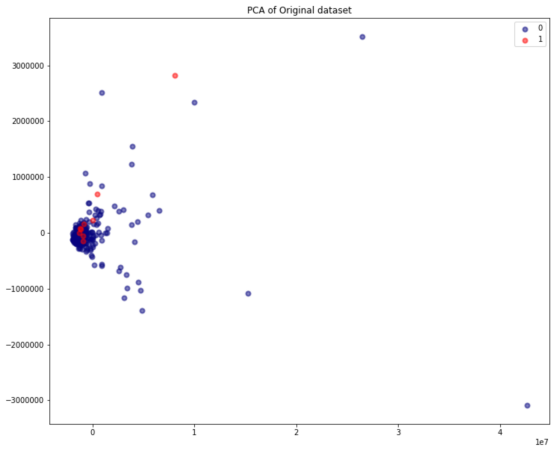

Figure 9: Pre-SMOTE Data

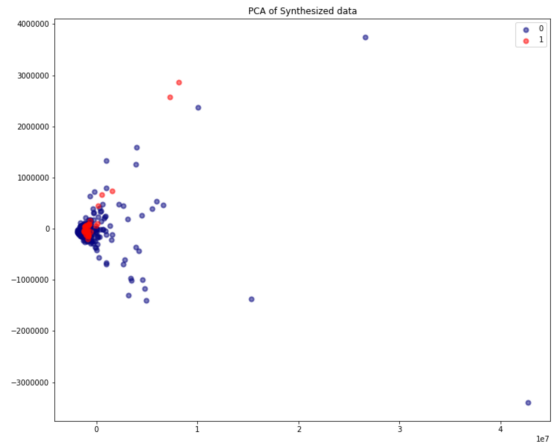

Figure 10: SMOTE Data

Implementation

Majority of the algorithms and their variations were implemented using off the shelf packages. For SVM and Random Forest, we used the Scikit Learn Python package[27], with the exception of F-score implementation for feature selection, which we implemented on our own. Majority of the Neural Network implementation was done using the CPU version of the Keras library [**chollet2015keras**]. All the algorithms were run on both versions of data, the original and the oversampled synthetic data. Since our aim was to assess the working of various algorithms on the chosen data, we evaluated each experiment on 3 metrics, namely, Accuracy score, area under the Receiver Operating Characteristic curve (ROC-AUC) and F_1-score. A brief explanation for each is provided in the next section.

Evaluation Metrics

Accuracy

Classification accuracy represents the number of correct predictions made divided by the total number of predictions. Although accuracy is the simplest and most common objective function for most binary classification, it is not the best metric when there is class imbalance. However, one major drawback of accuracy is that we assign equal weightage to both, false positives and false negatives. Because of the highly skewed nature of our dataset labels, even if the model predicted a single class for every instance possible, the accuracy would be 90

ROC-AUC

Area under the Receiver Operating Characteristic curve, is one of the best metrics available binary classification of imbalanced dataset. When the goal is to assess the accuracy of a classifier independent of any threshold, ROC analysis must be used. The ROC curve is obtained by plotting the True Positive Rate (TPR), against the False Positive Rate (FPR), at various thresholds. Hence, ROC evaluates the sensitivity as a function of fall-out. The Area under the curve (AUC) is equal to the probability that a classifier will rank a randomly

chosen positive instance higher than a randomly chosen negative one (assuming 'positive' ranks higher than 'negative'). Measuring the area under the ROC curve gives an indication of how well the leads are ranked. A perfect model will have ROC-AUC = 1, and a random guess model should have ROC-AUC= 0.5.

F_1-score

F_1-score is the weighted harmonic mean of precision and recall, where precision and recall are defined as:

$$\text{precision} = \frac{\text{True Positives}}{\text{True Positives +False Positives}}$$

$$\text{recall} = \frac{\text{True Positives}}{\text{True Positives +False Negatives}}$$

F_1 score is given by:

$$F_1 = \frac{1}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

A precision of 1 means that we have no false positives. A recall of 1 means that we have no false negatives. It is possible to have high precision and low recall, or vice-versa for classifiers, and proves to be a very strong analysis metric for an imbalanced dataset. The F_1-score reaches its best value at 1 (perfect precision and recall) and worst at 0.

Results & Analysis

We ran multiple implementation of all our algorithms on both the datasets, the original and the oversampled synthetic dataset. The results for each are shown in the table below:

Model	ROC	F-score	Accuracy
SVM + GS (original data)	0.55	.05	.85
SVM + GS (synthetic data)	0.98	.92	.98
FS + SVM + GS (original data)	.80	.20	.67
FS + SVM + GS (synthetic data)	.83	.92	.65
Ensemble SVM (original data)	N/A	N/A	.95
Ensemble SVM (synthetic data)	.98	.75	.96
RF + GS (original data)	0.5440	0.4895	0.9590
Weighted RF (original data)	0.5988	0.4895	0.9590
Balanced RF (original data)	0.5679	0.4822	0.9317
RF + GS (synthetic data)	0.9638	0.8179	0.9531
Weighted RF (synthetic data)	0.9876	0.8579	0.9617
Balanced RF (synthetic data)	0.9309	0.8303	0.9574
ANN - 1 layer (original data)	N/A	N/A	.93
Ensemble ANN - 3 layers (synthetic data)	.94	.84	.99

For the initial implementation, we trained SVM and RF with Grid Search and Cross Validation techniques, and a single layer network for the ANN on the original dataset. Although the accuracy scores were good, they were deceptively so due to the class imbalance in the data, and were akin to guessing a loan was approved, regardless of the features. These initial results led us to the conclusion that:

1. Different approaches were needed
2. Synthetic data would help to balance the classes

Once we generated synthetic data, one observation we made was that with a 5-fold cross validation, models had much better better results across the models. However for ANN the scores were good initially on the first cross-validation fold, and got progressively worse in later validation steps. This property generalized across multiple random splits. We leveraged this observation to conclude that instead of cross validating a single model K times, an ensemble network would work better by creating a new model with each fold. After experimenting with several different configurations, we had the best success with an ensemble of 5, 3 layer networks trained with a batch size 32 for 100 epochs. Our classifier then predicted based on a weighted average, normalized according to each model's training accuracy. After training the final model on 70% of the synthetic data, we tested on both the held out 30 percent of the synthetic data, and on the entire original dataset. While the results on the original data may not be statistically admissible, due to the fact that the model was trained synthetic data derived from it, the results on the held out data do suggest that there is a use case for neural networks outside of massive datasets, and that they can be used effectively in ensembles.

A similar approach was followed for Support Vector Machine. Looking at the results of the SVM + GS on synthetic data, we'd imagine we've got really good results, but it also seems like a case of overfitting. To evaluate further and we use 2 different approaches, feature selection with F-score and Ensemble SVM. Basing the success of Ensemble networks in Neural Networks, we also tried an SVM Ensemble network. It fails on the original data, and only predicts a single class for all permutation of the parameters. However, feature selection approach gave surprising results. Usually feature selection is performed to reduce dimensionality, but in this case it's that of eliminating noisy features. The FS + SVM + GS model on synthetic data has low average accuracy, but the other metrics are really high. Despite losing information by dropping features, the F1-score and the ROC-AUC metrics improve. The model predicts a balanced number of true positives and true negatives. Overall this seems like a very good model for the chosen dataset.

Alternatively, from a Random Forest point of view, we can observe that the RF's performance on accuracy is almost the same on all models. But as we already know accuracy is not the accurate measure for class imbalance problems. Even if we predicted all 0's for the data set, the accuracy would be 90%. On the other hand, ROC provides a better understanding of the model's performance. Weighted RF performs the best with higher accuracy. This is because the model penalizes the misclassifications on the minority class heavily. In the original data we see that the ROC and F1-scores are not good, in fact the results show that half of the classifications have false positives and false negatives. One reason original data does not perform could be due to the very low minority class data, roughly 2%. But among all the models on original data, weighted random forests gives the best result. This more evident in the results on synthesized data.

Synthesized data, as seen provides significantly better ROC and F-score. We observe that the F-score, accuracy and ROC is the highest for model with best parameters and added class weight. The ROC score is 98.7% which means there are only 1.3% of samples classified with false positives and false neg-

atives. Balanced RF's, using over sampling also achieves a good measure of ROC. Thus, we can conclude that weighted RFs work the best with imbalanced data sets.

References

[1] Lyn C. Thomas. "A survey of credit and behavioural scoring: forecasting financial risk of lending to consumers". In: *International Journal of Forecasting* 16.2 (2000), 149?172. DOI: 10.1016/s0169-2070(00)00034-0. URL: https://www.sciencedirect.com/science/article/pii/S0169207000000340.

[2] Milad Malekipirbazari and Vural Aksakalli. "Risk assessment in social lending via random forests". In: *Expert Systems with Applications* 42.10 (June 2015), pp. 4621–4631. DOI: 10.1016/j.eswa.2015.02.001. URL: http://dx.doi.org/10.1016/j.eswa.2015.02.001.

[3] B LARIVIERE and D VANDENPOEL. "Predicting customer retention and profitability by using random forests and regression forests techniques". In: *Expert Systems with Applications* 29.2 (Aug. 2005), pp. 472–484. DOI: 10.1016/j.eswa.2005.04.043. URL: http://dx.doi.org/10.1016/j.eswa.2005.04.043.

[4] A.f. Atiya. "Bankruptcy prediction for credit risk using neural networks: A survey and new results". In: *IEEE Transactions on Neural Networks* 12.4 (2001), 929?935. DOI: 10.1109/72.935101.

[5] Ajay Byanjankar, Markku Heikkila, and Jozsef Mezei. "Predicting Credit Risk in Peer-to-Peer Lending: A Neural Network Approach". In: *2015 IEEE Symposium Series on Computational Intelligence* (2015). DOI: 10.1109/ssci.2015.109.

[6] Brock. *Predicting Customer Behavior: How we use Machine Learning to Identify Paying Customers before they Subscribe*. URL: https://www.strong.io/blog/predicting-customer-behavior-machine-learning-to-identify-paying-customers.

[7] Zhi-Hua Zhou, Jianxin Wu, and Wei Tang. "Ensembling neural networks: Many could be better than all". In: *Artificial Intelligence* 137.1-2 (2002), pp. 239–263. DOI: 10.1016/s0004-3702(02)00190-x.

[8] Håvard Kvamme et al. "Predicting mortgage default using convolutional neural networks". In: *Expert Systems with Applications* 102 (2018), pp. 207–217. DOI: 10.1016/j.eswa.2018.02.029.

[9] Cheng-Lung Huang, Mu-Chen Chen, and Chieh-Jen Wang. "Credit scoring with a data mining approach based on support vector machines". In: *Expert Systems with Applications* 33.4 (2007), pp. 847–856. DOI: 10.1016/j.eswa.2006.07.007.

[10] Vladimir N. Vapnik. "Direct Methods in Statistical Learning Theory". In: *The Nature of Statistical Learning Theory* (2000), pp. 225–265. DOI: 10.1007/978-1-4757-3264-1-8.

[11] Benlan He et al. "Prediction of Customer Attrition of Commercial Banks based on SVM Model". In: *Procedia Computer Science* 31 (2014), pp. 423–430. DOI: 10.1016/j.procs.2014.05.286.

[12] Chih-Wei Hsu and Chih-Jen Lin. "A comparison of methods for multiclass support vector machines". In: *IEEE Transactions on Neural Networks* 13.2 (2002), pp. 415–425. DOI: 10.1109/72.991427.

[13] Platt, John and Cristianini, Nello and Shawe-Taylor, John. "Large Margin DAG's for Multiclass Classification". In: *Proc. Advances in Neural Information Processing Systems 12*. Jan. 1999. URL: https://www.microsoft.com/en-us/research/publication/large-margin-dags-for-multiclass-classification/.

[14] Hongzhuan Zhao et al. "A Multi-Classification Method of Improved SVM-based Information Fusion for Traffic Parameters Forecasting". In: *PROMET - TrafficandTransportation* 28.2 (Apr. 2016). DOI: 10.7307/ptt.v28i2.1643. URL: http://dx.doi.org/10.7307/ptt.v28i2.1643.

[15] Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (Oct. 2001), pp. 5–32. ISSN: 1573-0565. DOI: 10.1023/A:1010933404324. URL: https://doi.org/10.1023/A:1010933404324.

[16] Warren S. Mcculloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The Bulletin of Mathematical Biophysics* 5.4 (1943), pp. 115–133. DOI: 10.1007/bf02478259.

[17] George F. Luger. *Artificial intelligence: structures and strategies for complex problem solving*. Pearson Addison-Wesley, 2009.

[18] "Perceptron Algorithm, 1959; Rosenblatt". In: *SpringerReference* (). DOI: 10.1007/springerreference-57806.

[19] B. Widrow and M. E. Hoff. "Adaptive Switching Circuits". In: (1960). DOI: 10.21236/ad0241531.

[20] URL: http://cs231n.github.io/convolutional-networks/.

[21] John Wang and Alan Oppenheim. "The Pitfalls of Knowledge Discovery in Databases and Data Mining". In: *Data Mining* (). DOI: 10.4018/9781591400516.ch009.

[22] Hsu, C. W., Chang, C. C., and Lin, C. J. (2003). *A practical guide to support vector classification*. URL: http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf.

[23] Yi-Wei Chen and Chih-Jen Lin. "Combining SVMs with Various Feature Selection Strategies". In: *Feature Extraction Studies in Fuzziness and Soft Computing* (), pp. 315–324. DOI: 10.1007/978-3-540-35488-8-13.

[24] L.K. Hansen and P. Salamon. “Neural network ensembles”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12.10 (1990), pp. 993–1001. DOI: 10.1109/34.58871. URL: http://dx.doi.org/10.1109/34.58871.

[25] Chao Chen, Andy Liaw, and Leo Breiman. “Using random forest to learn imbalanced data”. In: *University of California, Berkeley* 110 (2004), pp. 1–12.

[26] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2014). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

[27] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.