CS 475 Machine Learning: Final Project
Dual-Form SVM for Predicting Loan Defaults

Kevin Rowland
Johns Hopkins University
3400 N. Charles St.
Baltimore, MD 21218, USA
krowlan3@jhu.edu

Edward Schembor
Johns Hopkins University
3400 N. Charles St.
Baltimore, MD 21218, USA
eschemb1@jhu.edu

Abstract

Machine learning methods have become increasingly present in the financial industry in the past few years, from automated equity trading to generating insurance policies. The goal of our project lies in this financial vein: we utilize machine learning methods in order to predict whether a borrower will default on a loan. A Support Vector Machine is used to construct a decision boundary between loans that are predicted to be fully paid off and loans that are predicted to default.

1 Introduction

Lending Club is a peer-to-peer marketplace for loans. Borrowers are asked to provide personal financial data in order for Lending Club to reject or accept the loan proposal. In the event of acceptance, Lending Club attaches a grade to the loan that determines the interest rate. Given historical data of loans that passed Lending Club's initial acceptance test, this project aims to predict which borrowers will default on their loans. Historical data sets, provided by Lending Club for public analysis, are used to train the SVM model.

A Support Vector Machine is a supervised learning model used for binary classification. Training data is formatted as a list of d real-valued features. An SVM constructs a hyperplane in d-dimensional space that optimally separates the data into two classes. A soft-margin SVM allows for "slack," whereby some training data is allowed to lie inside the margin or on the wrong side of the hyperplane. In order to construct the hyperplane, this project solves the dual form of the optimization problem.

An SVM was chosen for our project because the goal is to predict whether a loan falls in one of two categories (default or paid off). The ability to use slack variables fit well with the non-separable nature of our data.

Existing literature includes previous attempts to apply machine learning techniques to predicting loan defaults (Ramiah, Tsai, Singh, 2014; Wu, 2014). Other analysts have approached the problem from a data science perspective - revealing patterns in the data but not using machine learning to predict outcomes (Davenport, 2013).

2 Support Vector Machines

The optimization problem that a Support Vector Machine attempts to solve can be formulated in two ways: the primal form or dual form. The primal form optimizes its objective function by simultaneously minimizing the prediction error and the size of the weights vector used to make predictions. The dual form of the optimization problem attempts to maximize the following objective.

$$\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j \tag{1}$$

With the following constraints:

$$0 \leq \alpha_i \leq U \quad \forall i \tag{2}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{3}$$

Where the training examples associated with non-zero α_i are called support vectors.

In order to learn the support vectors, we implemented a dual coordinate descent method described in Hsieh et al. (2014) with an L1 loss function of the form $max(1 - y_i \mathbf{w}^T \mathbf{x}, 0)$. Dual coordinate descent is an algorithm that iterates over each training example, optimizing the objective in equation 1 with $i = j$. Essentially, the algorithm optimizes the objective for one example while ignoring all other examples. The algorithm reaches an ϵ-accurate solution in $log(1/\epsilon)$ iterations. ϵ-accuracy implies that equation 1 is within ϵ of its maximum value. This means that with I as the number of iterations and ϵ as the achieved accuracy:

$$\epsilon = e^{-I} \tag{4}$$

Predictions using the support vectors are done by computing the following, with $\mathbf{x}$ as the sample instance to be predicted:

$$\sum_{A} \alpha_i y_i \mathbf{x}_i \mathbf{x} \tag{5}$$

Where A is the set of all support vectors, i.e. all training points x_i with non-zero α_i

The algorithm was implemented in Java using the wrapper classes provided for the course (Instance, FeatureVector, Label, etc.). A detailed description of the algorithm implemented can be found in Hsieh et al. (2014).

3 Methods

3.1 Data Preprocessing

In order to convert the data from Lending Club's raw .csv format into a format which the SVM algorithm could interpret, a python script was used. Most of the data simply required a re-formatting. However, some fields did require more manipulation. For example, one data field holds a string representing the current status of the loan. This string can take values "Charged Off", "Current", or "Fully Paid." The python script was used to locate these strings and convert them to the labels.

In order to quantify the field which contained raw text that explains the reason for a loan (i.e. "I plan on combining three large interest bills together and freeing up some extra each month to pay toward other bills. I've always been a good payor [sic] but have found myself needing to make adjustments to my budget due to a medical scare. My job is very stable, I love it."), the TextBlob python text processing library was used to perform sentiment analysis. Each loan reason was classified as having "Positive", "Negative", or "Neutral" sentiment, then given a value of 1, -1, or 0, respectively.

Due to the dataset from Lending Club being heavily skewed towards paid off loans (only 20% of loans defaulted), the training data posed some problems for soft-margin SVMs. Wu and Chang (2003) hypothesized that heavily skewed data will lead to an imbalanced support vector ratio, thus favoring prediction of one class over another despite the constraint in equation 3 that the set of the support vectors on each side of the boundary have the same weight. Akbani et al (2004) refute this point and posit that a soft-margin SVM with heavily skewed data will tend to make the margin as large as possible while still keeping the error low due to the large amount of correctly classified positive examples. It was therefore decided to balance the data so that our soft-margin SVM could learn a boundary that was not affected by heavy skew. Balancing was done by trimming the training data so that it had the same number of defaults as it did non-defaults, an even 50-50 split.

3.2 Iterations

An initial value for the number of iterations for which to run our algorithm was chosen arbitrarily as 10. This was mainly for the purpose of keeping training time as short as possible. However, according to equation 4, 10 iterations will lead to an accuracy of $e^{-10} = 0.000454$, which is acceptable for our testing purposes.

3.3 Bias and Margin

In order to optimally separate the two classes of data, it was necessary to add an extra dimension to our feature space in the form of a bias term b. Because our data is not centered (most features are strictly positive valued) adding a bias term allows the hyperplane to better fit the data by allowing it to pass through a point other than the origin. In the process of determining a bias term, it was decided to choose b from $[\|\mathbf{x}\|, 10\|\mathbf{x}\|]$ (VLFeat,

2013), where $\|\mathbf{x}\|$ is the average Euclidean norm of our training points. It was also necessary to choose a value for U, the upper bound on α_i from 2, which corresponds to the Lagrange multiplier on the error term in the primal formulation of the SVM objective. Increasing U gives more value to minimizing errors and less value to maximizing the margin, while decreasing U does the opposite. Some manual experimentation was done by varying U until our number of support vectors decreased and our accuracy increased to reasonable values. It was then determined empirically that U should lie in the range $[10^{-10}, 10^{-11}]$. With an average Euclidean norm of approximately 77,000, a grid search was performed on values of b and U. A 5x5 sample of the full grid is shown in Table 1 below. The highest training accuracy achieved across the grid is underlined in the table at $(b = 308, U = 3E - 11)$.

		b (x 1,000)				
		154	**231**	**308**	**385**	**462**
U (E-11)	**1**	.570	.563	.571	.571	.573
	2	.575	.575	.582	.587	.571
	3	.576	.573	<u>.588</u>	.583	.582
	4	.579	.587	.582	.586	.584
	5	.581	.587	.584	.585	.583

Table 1: Partial Grid Search on Original Data

After using grid search to find an optimal pair of b and U, attempts were made to normalize the data. Without normalization, features whose values are naturally high (e.g. salary) will outweigh features with naturally low values (e.g. loan term (months)). Normalization was performed by scaling each feature to the range [0,1]. After normalization, we again performed a grid search on values of b and U. This time, b was taken from $\{1.7, 3.4, ..., 17\}$ and U was taken from $\{0.2, 0.4, ..., 2.0\}$. Again, the highest training accuracies achieved across the grid are underlined in the table at $(b = 1.7, U = 1.2)$ and $(b = 3.4, U = 0.8)$.

		b				
		1.7	**3.4**	**5.1**	**6.8**	**8.5**
U	**.4**	.593	.593	.597	.524	.503
	.6	.526	.598	.527	.505	.501
	.8	.588	<u>.602</u>	.525	.503	.501
	1.0	.529	.540	.508	.502	.501
	1.2	<u>.602</u>	.529	.505	.502	.501

Table 2: Partial Grid Search on Normalized Data

4 Results

Before training our model on the data from Lending Club, it was necessary to verify that our algorithm worked reasonably well on data sets that were known to be separable. The model as therefore run on the synthetic "easy" data provided for the course by Dr. Shpitser. This quick test resulted in a training accuracy of 94%, higher than the 90% accuracy obtained by PEGASOS.

After validating our model with the synthetic data, the algorithm was ready to run on our pre-processed loan data. Along with training and testing accuracy, two additional metrics were calculated: recall and precision. First, let us define the "positive" side of the hyperplane as the side on which loans are predicted to be fully paid off. Consequently, the "negative" side of the hyperplane will refer to the side on which loans are predicted to default. The first metric, recall, refers to the number of fully paid off loans that were correctly classified on the positive side of the hyperplane. The second metric, precision, refers to the number of test points on the positive side of the hyperplane that were correctly classified. It can be thought of as an accuracy measure restricted only to the positive side of the separating hyperplane. A higher precision equates to a lower number of false positives, i.e. a lower number of loans that are predicted to be fully paid off but actually default. Recall and precision can be calculated by keeping track of true positive (TP), false positive (FP), true negative (TN) and false negative (FN) classifications:

$$recall = TP/(TP + FN) \tag{6}$$

$$precision = TP/(TP + FP) \tag{7}$$

For the purposes of our project, a higher precision was preferred over higher accuracy or recall. If a creditor were to issue loans based on a positive

prediction from our SVM, we seek to minimize the number of times those loans will default and therefore maximize the return for said creditor. This equates to minimizing the number of false positives and therefore maximizing the precision.

An example of the output from our model is shown below:

```
BIAS      : 308000.0
U         : 9.0E-11
Train     : 0.5805464480874317
Recall    : 0.5616208975217682
Precision: 0.7332750327940534
```

4.1 Results with Original Data

After running a grid search on the training data and generating 100 results, some basic identification was done to determine the parameters that gave the best values for each of our metrics. As mentioned in the previous section, our priority was to maximize precision. However, our grid search also revealed the set of parameters that maximize training accuracy and recall. After developing a model based on the training set, a good test of the sensibility of the model is to run it on a separate test data set. Therefore, the rows below were chosen by identifying the training parameters that generated the highest accuracy, highest recall, and highest precision, respectively, and then applying those parameters to the test data set.

U (E-11)	Bias (E3)	Train Acc.	Test Acc.	Recall	Precision
2.0	385	.587	.546	.527	.888
2.0	770	.551	.557	.538	.809
6.0	770	.511	.546	.527	.891

Table 3: Results with Original Data

4.2 Results with Normalized Data

Again, the rows below were chosen by obtaining the parameters that maximized training accuracy, precision, and recall. However, the same pair of parameters generated both the highest accuracy and the highest recall, so those two rows were reduced to just the top row.

U	Bias	Train Acc.	Test Acc.	Recall	Precision
1.2	1.7	.602	.499	.499	1.00
0.8	15.3	.500	.501	.800	.003
.6	1.7	.586	.499	.499	1.00

Table 4: Results with Normalized Data

4.3 Results with Home Ownership Models

One feature that was thought to provide some insight into the riskiness of a loan was home ownership. However, the corresponding field in the raw data contained a string giving the home-ownership status of the borrower as a string. The possible values of the string were “MORTGAGE”, “RENT”, “OWN”, and “OTHER”. Instead of spending time trying to translate those strings into representative real-valued numbers for our model to train with, we decided to eliminate the feature from our training altogether. After obtaining reasonable results with our original and normalized data sets (see “Comparison to Related Literature” - Section 4.5), we decided to pursue an analysis of the effects of home ownership on our data. To do this, we split our normalized data into three buckets, one each for “MORTGAGE”, “RENT”, and “OWN”. We then trained our model separately on each of these buckets. Results for the separate models are shown in the tables below.

First, the model was run only on loans in which the borrower has a mortgage on his house. Again, the rows below were chosen by identifying the training parameters that generated the highest accuracy, highest recall, and highest precision, and then applying those parameters on the test data set.

U	Bias	Train Acc.	Test Acc.	Recall	Precision
0.2	8.5	.621	.593	.571	.867
0.4	6.8	.485	.537	.790	.148
0.4	1.7	.541	.519	.519	1.00

Table 5: Results with “MORTGAGE” Data

The model was then run on loans in which the borrower owns a house.

U	Bias	Train Acc.	Test Acc.	Recall	Precision
0.2	3.4	.624	.605	.553	.816
0.2	1.7	.570	.468	.468	1.00

Table 6: Results with "OWN" Data

Finally, the model was run on loans in which the borrower rents.

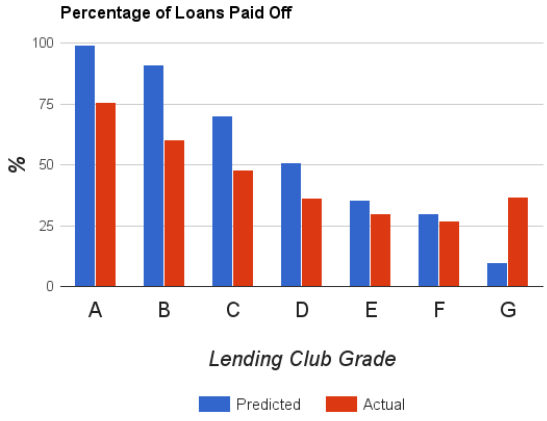
U	Bias	Train Acc.	Test Acc.	Recall	Precision
0.2	1.7	.582	.488	.488	1.00
0.4	8.5	.549	.511	.500	0.00
0.4	1.7	.582	.488	.488	1.00

Table 7: Results with "RENT" Data

One can see how the accuracy tends to increase when we remove the variability in home ownership.

4.4 Comparison to Lending Club Grades

After determining that a potential borrower qualifies for a loan from Lending Club members, the data provided by the borrower is used by LC to give the loan a grade. Grades are lettered A-G, with subgrades within each grade numbered from 1-5. In order to give a more detailed view of the relationship between our prediction and the real-world risk profile of the loan, we compared our predictions with the letter grade given by LC. To do this, we kept track of how many loans from our input data set were given for each grade. We also kept track of how many loans were predicted by our model to be paid off for each grade. Below is a histogram of our predictions for each grade bucket.



The above histogram shows a strong downward trend in the predicted rate of paid-off loans. This shows that the results of our algorithm are likely similar to the results of the algorithm used by Lending Club to classify their loans. The higher the grade given by Lending Club, the higher rate of success predicted by our SVM. It seems that certain combinations of features lead to certain rates of success and that our SVM has exposed that pattern. In addition, our predicted rates of success by grade correlate strongly with the real life rates of success.

4.5 Comparison to Related Literature

In order to place our results in the context of related literature and to ensure that our accuracy was reasonable given our data, results from similar projects were used for comparison. Ramiah et al. (2014) ran various machine learning algorithms on data from Lending Club, including an SVM model trained using the popular library libSVM. The highest precision obtained by their SVM was 93.7%, which came at the cost of a recall of only 10.7%. The rest of their tests yielded precision values in between 81% and 88%.

Data generated by our model yields similar results. Our best training accuracy of 62.4% came on normalized data limited to borrowers who owned their own houses ($b = 3.4, U = 0.2$). The associated testing accuracy for this model was 60.5%, with a recall of 55.3% and a precision of 81.6%. A similar set of parameters ($b = 1.7, U = 0.2$) led to a training accuracy of 57% and an astonishing precision of 100%. The associated accuracy and recall fell to 46.8% and 46.8% respectively.

4.6 Comparison to Proposal

The primary goal described in our proposal was the implementation of an SVM model with soft-margins trained and tested on the Lending Club data. This goal was successfully completed, the test results of which can be seen above.

A secondary goal was to create separate models based on each class in "home ownership" to see if we could improve performance by creating specific models. This goal was accomplished and results were analyzed in Section 4.3.

The main "reach goal" was to compare the predictions of the SVM to the grades assigned to the loans

by Lending Club. This goal was also accomplished. The performance of our SVM on each letter grade is shown in Section 4.4.

5 Future Work

There exist many methods that can be used to improve the performance of our model. One such method would be the application of the kernel trick. The kernel trick would allow projection of data into higher dimensions in order to better separate the data. Many simple kernels can be tested, including the Radial Basis Function kernel or the polynomial kernel. Due to the non-separability of our data, application of the kernel trick might have led to increases in the performance of our model.

Another improvement that could be implemented is to make better use of sentiment analysis. Currently, our pre-processing application of the TextBlob library is limited to returning three discrete values (-1, 0, 1). Performance may increase with a more continuous set of values.

Additionally, use of the original data preserved the difference in scale between different features. For example, the salary feature tended to take values on the order of 10^4, while the loan term feature is on the order of 10. This disparity in scale gives implicit weight to larger features. Similarly, by normalizing the data, all of the feature values are on the same order of magnitude, which results in an equal distribution of weight across features.

These assumptions generally do not hold. For example, the salary of a borrower may have a much higher correlation with the default rate, while the term length may not be a strong predictor. One way to apply a better distribution of weights on our features, a prior could be placed on the weights vector before training. Similarly, features could be manually scaled to an order of magnitude that reflects their predictive power.

Finally, in future works, a focus could be placed on maximizing precision. As explained in section 4, a higher precision would yield better results for a potential lender.

References

Akbani, R., Kwek, S., Japkowicz, N. (2004). *Applying Support Vector Machines to Imbalanced Datasets.* 5th European Conference on Machine Learning, Pisa, Italy

Wu, G. and Chang, E. (2003). *Class-Boundary Alignment for Imbalanced Dataset Learning.* ICML 2003 Workshop on Learning from Imbalanced Data Sets II, Washington, DC.

Ramiah, S., Singh, S., Tsai, K. (2014). *Peer Lending Risk Predictor* Stanford CS229 Machine Learning

Wu, J. (2014). *Loan Default Prediction Using Lending Club Data* NYU

Davenport, K. (2013). *Gradient Boosting: Analysis of LendingClubs Data*

VLFeat. (2015). *C Documentation: SVM Fundamentals* Retrieved from http://www.vlfeat.org/api/svm-fundamentals.html

Lending Club. (2015) *Loan Data 2007-2011* Retrieved from https://www.lendingclub.com/info/download-data.action.

Ng, A. (2015) *CS229 Lecture Notes. Part V: Support Vector Machines* [pdf] Retrieved from http://cs229.stanford.edu/notes/cs229-notes3.pdf