Stanford University

MS&E 448

Algorithmic Trading and Big Financial Data

P2P Loan Selection

Team Members:
Andy Feis
Viraj Mehta
Scott Morris
John Solitario
Cameron Van de Graaf

June 4, 2016

Abstract

The peer-to-peer lending industry has grown significantly since its inception in 2007. With billions in annual loans, there are significant opportunities to capitalize on this alternative investment instrument. We have developed a sophisticated investment strategy that utilizes Lending Club Corporation's massive historical datasets to understand which features best predict someone's probability of default.

Using these characteristics, we built and tested numerous machine learning models, including logistic regression, SVM, random forest and other ensemble methods. Our final model with the most accurate results used linear discriminant analysis (LDA). These models outputted an implied probability of default, which we converted to a continuous price using our pricing model. We were able to deliver 2-3% increases in loan returns over the average basket loans provided by Lending Club. This yielded a tentative Sharpe ratio of 3.41.

We then built an execution platform that uses Python scripts to communicate with Lending Club's API. The platform allows for automated feature processing, analysis, and loan acquisition.

1 Project Background

Peer-to-peer lending, often abbreviated P2P lending, is a method of debt financing that enables individuals to borrow and lend money - without official intermediary institutions, such as banks. The modern P2P lending industry began in the United States in 2006 with the birth of Prosper, followed by Lending Club and other lending platforms thereafter [4]. As of March 2016, Lending Club is the largest peer-to-peer lender in the United States with over $18 billion in total loan issuance to date [3]. Due to data access as well as loan data sample size, we chose to focus on Lending Club specifically as a means of understanding the P2P lending market.

Given the turbulence in the market, many investors have looked towards P2P lending as an alternative investment instrument to achieve returns. The way Lending Club works is simple: an individual or business seeking a loan

completes an application with numerous predictive characteristics to Lending Club. Lending Club then uses a proprietary algorithm to approve loans and place the approved loans into "buckets." These buckets range from A1, A2, A3...G4, and G5. Essentially, Lending Club assesses a loan's risk of default and attempts to place said loan into a corresponding bucket, with A1 being the most secure loans and G5 being the most risky. Additionally, each bucket has a corresponding interest rate. Naturally, the A1 loans (which are safest in Lending Club's eyes) have the lowest interest rate, 5.32%. A2 loans have a slightly higher rate of 6.49%, and so on until G5 loans, which have a 30.99% interest rate.

As an investor, you can either choose to invest in a "basket" of Lending Club loans or individual loans. If you invest in the basket, Lending Club takes your principle and divides it up across numerous loans, often filling only a small part of a loan, to increase diversification and minimize the large downside risk of a single loan defaulting. Alternatively, you can pick and choose loans to invest in, which could be beneficial if you believe you have found a loan whose risk has been assessed incorrectly.

2 Strategy Description

The discrete bucket model that Lending Club and other peer institutions use was quite intriguing to our group. While we completely understood the need for varying interest rates depending on the risk of loan default, it seemed quite arbitrary that there are lines drawn between buckets. Clearly not every loan in a bucket, take A2 for example, has exactly the same risk, which is why assigning them the same rate of interest doesn't quite make sense. Moreover, the very best loans in A2 and the very worst loans in A1 are likely quite similar in risk; however, Lending Club has placed them into different buckets. This led our group to question why, if two loans are similar, can one have an interest rate of 5.32% while the other has a 6.49% rate for nearly the same risk.

We believed that if we could create a continuous model for assessing loan risk, we could identify the safest loans in each bucket. Thus, we could get the same interest rate while decreasing our default risk, increasing returns over the basket approach provided by Lending Club.

This strategy required three primary phases: feature selection, modeling default risk/building a pricing model, and execution. Fortunately, Lending Club publishes all their historical loan data along with the 100+ characteristics about the party receiving the loan and the loan status. The first step of the strategy, feature selection, involved identifying which of the provided characteristics were most indicative of whether or not a loan would be paid in full or charged off. The next step, creating the model, required taking those features and using multiple machine learning techniques to predict a loan's risk of default given the features selected in phase one. Lastly, finding and acquiring the loans in real-time requires an execution strategy for this approach to be scalable. Thus, the last phase of the project was to build a platform for executing our model and acquiring the newly discovered desirable loans.

3 Data Cleaning and Processing

Lending Club provides five online, open-access datasets for accepted loans from 2007-2011, 2012-2013, 2014, 2015, and 2016 Q1 in comma-separated-values (CSV) format. They also provide four additional datasets for declined loans from 2007-2011, 2012-2013, 2014, 2015, and 2016 Q1. Each accepted loan dataset has 112 variable fields; however, for the older datasets approximately 60 of these variable fields were left empty, narrowing down the number of possible features to 62. We then narrowed down the number of possible features to 22 because many of the variables are unrelated to loan quality.

In order to reduce the number of features down from 22, we used a series of filter and wrapper methods for the feature selection process. The filter method, also known as variable ranking, is a preprocessing step, independent of the choice of the predictor. It involves computing the Pearson correlation coefficients between all feature variables in order to assess linear dependencies between variables [5]. Variables then can be selected according to individual predictive power, using as a criterion the performance of a classifier built with a single variable. The correlation table for the final selected 11 variables is listed below.

	loan_status	loan_amnt	emp_length	home_ownership	annual_inc	verification_status	dti	inq_last_6mths	open_acc	revol_bal	revol_util	total_acc
loan_status	1	0.010718327	0.017272628	0.02697888	0.0439936	0.0037642	-0.034415383	-0.074169407	0.01649611	0.001625551	-0.086700536	0.036143098
loan_amnt	0.010718327	1	0.118774612	0.077193628	0.267839904	0.313338349	0.027853603	-0.005793888	0.173718863	0.310312711	0.039748872	0.233605029
emp_length	0.017272628	0.118774612	1	0.189125588	0.124791054	0.051026275	0.044429288	-0.002198094	0.094723646	0.144964143	0.01196939	0.200057498
home_ownership	0.02697888	0.077193628	0.189125588	1	0.118528702	0.011319636	-0.021719833	0.056484589	0.136529014	0.137277015	-0.105512739	0.227584342
annual_inc	0.0439936	0.267839904	0.124791054	0.118528702	1	0.11666984	-0.123348097	0.029051666	0.153243388	0.278156404	0.009516563	0.23259874
verification_status	0.0037642	0.313338349	0.051026275	0.011319636	0.11666984	1	0.046581309	-0.002697903	0.069319929	0.124962744	0.043002953	0.091164409
dti	-0.034415383	0.027853603	0.044429288	-0.021719833	-0.123348097	0.046581309	1	0.005389789	0.28513401	0.220606421	0.277286286	0.22189278
inq_last_6mths	-0.074169407	-0.005793888	-0.002198094	0.056484589	0.029051666	-0.002697903	0.005389789	1	0.098734557	-0.018427605	-0.068854616	0.112368473
open_acc	0.01649611	0.173718863	0.094723646	0.136529014	0.153243388	0.069319929	0.28513401	0.098734557	1	0.286992782	-0.091948665	0.689161665
revol_bal	0.001625551	0.310312711	0.144964143	0.137277015	0.278156404	0.124962744	0.220606421	-0.018427605	0.286992782	1	0.297092995	0.313374372
revol_util	-0.086700536	0.039748872	0.01196939	-0.105512739	0.009516563	0.043002953	0.277286286	-0.068854616	-0.091948665	0.297092995	1	-0.079115494
total_acc	0.036143098	0.233605029	0.200057498	0.227584342	0.23259874	0.091164409	0.22189278	0.112368473	0.689161665	0.313374372	-0.079115494	1

Table 1: Correlation table for predictors and response

These variable correlations can also be viewed visually by examining the normalized variable distributions (For the image Loan Amount, Annual Income, and Revolving Balance have been divided by 1000).

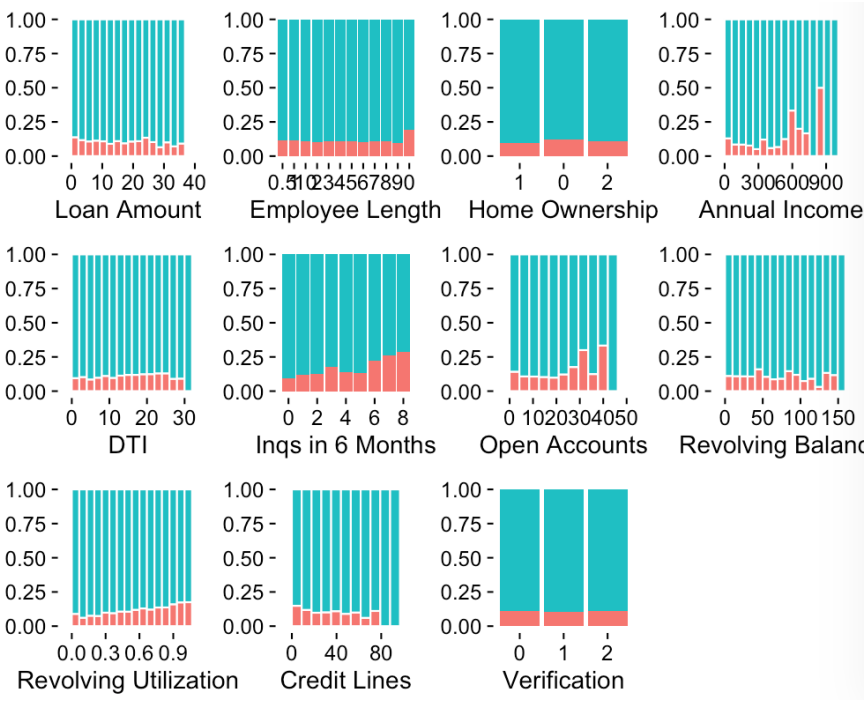

Figure 1: Normalized predictor distributions

After verifying variables through the filter method, we then further assessed predictability power through a wrapper method. Through a general framework, the wrapper method consists in using the prediction performance of a given learning machine to assess the relative usefulness of subsets of variables [7]. In this case, we used a logistic regression as the learning machine.

Each variable was used as the only feature of the logistic regression and then trained on the training portion of 2007-2011 dataset. The regression was then ran on the testing portion of the 2007-2011 dataset. In order to assess the variables predictability, the area under the curve (AUC) was calculated from the results of the testing set. The results for the final 11 variables are listed below.

Feature	Validation
loan_amnt	0.52393192
emp_length	0.51635724
home_ownership	0.50825862
annual_inc	0.54624598
verification_status	0.51720652
dti	0.52421064
inq_last_6mths	0.55482446
open_acc	0.51269613
revol_bal	0.50891253
revol_util	0.56982618
total_acc	0.52316278

Figure 2: AUC results

Listed are our model's 11 feature variables: loan amount, employment length, home ownership, annual income, verification status, debt-to-income ratio, inquiries in the last 6 months, open accounts, revolving balance, revolving utilization, and total accounts. Verification status indicates whether annual income numbers were verified by Lending Club, self-verified by the borrower, or not verified at all. Debt-to-income ratio is calculated using the borrower's total monthly debt payments plus the requested Lending Club loan, divided by the borrower's self-reported monthly income. Inquiries in the last 6 months represent the number of credit inquiries over that period. Open accounts indicate the number of open credit lines in the borrower's credit file. Revolving balance represents the total of all balances on all revolving charge accounts. Revolving utilization indicates the amount of credit the borrower is using relative to all available revolving credit. Total accounts represent the total number of credit lines currently in the borrower's credit file. The remaining feature variables are self-explanatory. Originally, we included the loan term variable in our model, but later decided to focus on only 36-month loans.

In order to use all of these features for our model, we had to convert several classification variables into numerical values. For home ownership, the sta-

tuses of Other, None and Rent were converted to 0, Mortgage was converted to 1, and Own was converted to 2. For Verification Status, Not Verified was converted to 0, Source Verified was converted to 1, and Verified was converted to 2. All of these conversions make safe assumptions about the weighting of each classification.

In order to train our model, we decided to first use the 2007-2011 dataset of accepted loans because at this point all of the loans in the dataset have either been charged off or fully paid. This limits any temporal issues and helps us avoid a biased sampling. In the end this gave us over 35,000 loans to examine.

4 Modeling

4.1 Overview

After data selection and cleaning, we evaluated a number of different machine learning techniques for their accuracy and stability in predicting loan default risk. All analyses were conducted in Python with the Numpy-Pandas-Scikit stack. For cross-validation and out-of-sample testing we used the Scikit cross-validation module along with the train-test split method. Our usual approach was to use 66% of data for training and hold the remainder out for testing. Since the two classes we wished to predict, defaulting vs. non-defaulting loans, were not balanced (default rate across all buckets was only 15%) standard classification algorithms were not uniformly successful out of the box. We therefore set specified probability cutoffs and compared that value with the implied probability of default generated by each model. For example, we might only decide to purchase a loan if the model predicted a $< 10\%$ probability of default compared to a median default rate of 15%. Moreover, we first ran through a broad range of algorithms on the full loan dataset without adjusting by bucket in order to gain an intuition for which models were likely to perform best, before moving forward with the most promising models to train and test on individual buckets. We will now provide an overview of some of the ML methods we tested.

4.2 Logistic Regression

The first algorithm we assessed was logistic regression. Logistic regression is a generalized linear model used for classification tasks. It works by fitting a series of coefficients corresponding to feature weightings to a sigmoid function. Coefficient optimization was performed using the coordinate ascent algorithm. When tested on our initial set of 8 features (loan amount, employment length, annual income, verification status, debt-to-income ratio, open accounts, revolving utilization, and total accounts), we achieved an average test accuracy of 88% when we set the probability cutoff at the median default rate of 85%. We were encouraged by these results, as it indicated that our features did contain predictive information.

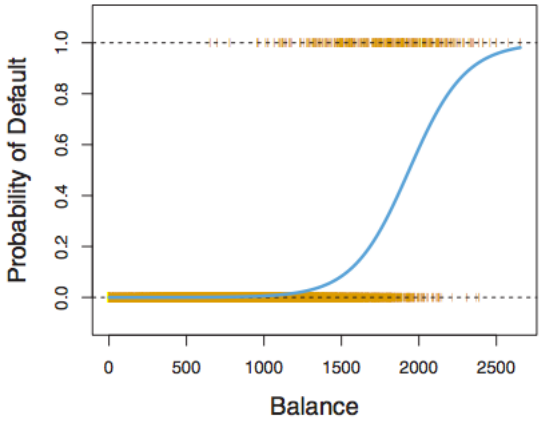

Figure 3: Example of logistic regression in the two-dimensional setting from [6]

4.3 Linear Discriminant Analysis

The second algorithm we tested was linear discriminant analysis (LDA), another linear classification method that assumes a Gaussian distribution on the underlying features. In LDA, the probability estimate of a point is derived from the distance of the point from the separating hyperplane [6]. Though often used for dimensionality reduction, we found that the LDA performed surprisingly well on our dataset. Again, testing on our initial set of 8 features,

we achieved an average test accuracy of 92% when we set the probability cut-off at the median default rate of 85%. We will go into more detail on LDA later, as it formed the core of our final model.

4.4 Support Vector Machines

We next attempted to use support vector machines with both linear and non-linear kernels. Support vector machines (SVM) have been used in the past for similar problems. While linear SVMs are widely used in classification and function by deriving two so-called support vectors which delineate the decision boundary. Unlike other linear models, SVM are only fit by the training points nearest to the decision boundary, operating from the principle that we should focus on the examples that are hardest to classify. While initial results with these models were promising, the best sustained performance we were able to achieve was an average test accuracy of 89% on a median default rate of 85% using a linear kernel. Non-linear kernels, including radial and polynomial variants, did not yield results above the median default rate. This seems to indicate, in accordance with our intuition, that many of our predictors varied fairly linear with the response.

4.5 Ensemble Methods

The final machine learning models we tested were of the ensemble variety, specifically a classification tree, random forest classifier, and AdaBoost classifier. Ensemble models are well-known as effective out-of-the-box ML algorithms that require little tuning of hyperparameters and are able to capture non-linear relationships between predictors and the response [6]. AdaBoost is a particularly widely used algorithm that works by fitting a series of weak learners that cumulatively provide superior predictive performance. When testing these models on the 2007-2011 dataset, we achieved best performance with the AdaBoost classifier, attaining a test accuracy of 91.7%, comparable to that of our LDA model. However, due to the noticeable increase in training time required for AdaBoost, we decided to proceed with our LDA model.

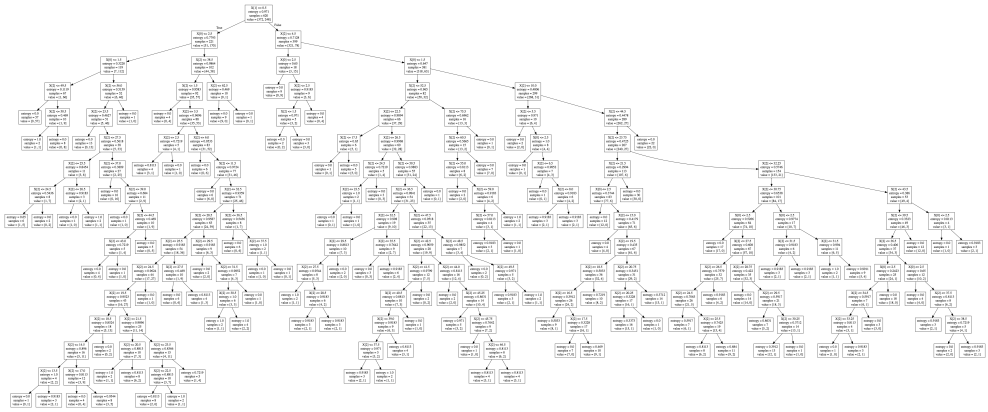

Figure 4: Portion of our classification tree

4.6 Final Model and Results

As stated earlier, the final model we used was a linear discriminant analysis. We first trained the model in the same manner as previous algorithms, employing a 66-33 train-test split. However, this time we trained and tested specifically within each loan bucket to see whether the model would achieve our goal of finer-grained loan sorting. We also ran the model repeatedly on a series of "confidence intervals," or the percentile cutoff for loan probability of default. Finally, for in-sample tests we ran 20 randomized trials to assess the consistency of results.

Below are a selection of results in chart format. The first chart demonstrates the predominantly upward sloping relationship between our degree of confidence in the quality of the loan (by percentile of probability of not defaulting) and the difference between default rate in our selected loans and the median default rate of the bucket. One slight caveat is that what we refer to as the "default rate differential" is only an approximation of the difference in default rate, as this differential actually measures the difference between the number of perfect loans (loans whose payments were always made on time, never late or in default) chosen by our model and the median. This simplifying assumption was made to ensure a two-class prediction problem, but has the effect of inflating some of the absolute numbers associated with our increase in predictive accuracy.

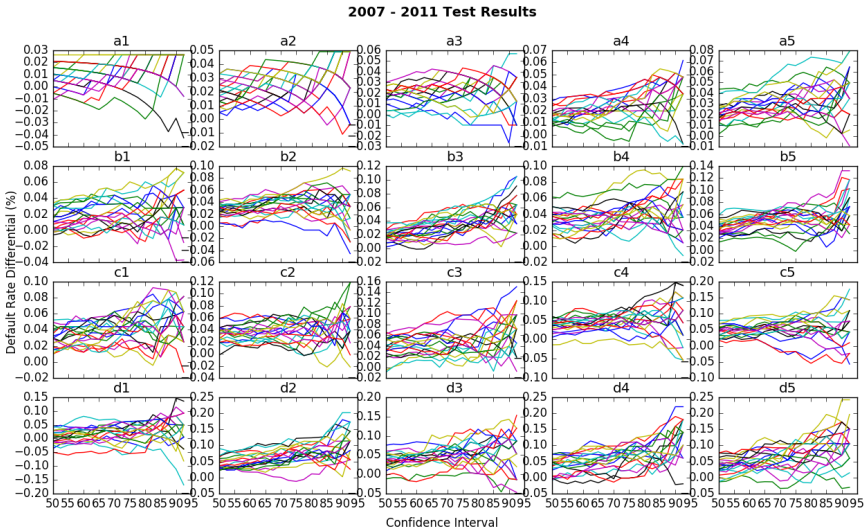

Figure 5: In-sample results in loan buckets A-D, with 20 random trials per bucket

The next chart shows the same results, but this time the differential is averaged across all of the buckets. This has the helpful effect of smoothing out some of the variance resulting from the relatively small sample sizes of loans selected, especially at higher confidence intervals in some of the more sparsely populated buckets. This chart clearly demonstrates the expected increase in predictive accuracy as model specificity increases. For instance, if we were to select only the top 25% of loans according to our model (75th confidence interval), we could expect a differential of about 3.5% between our selected loans and the median loan. This translates into a roughly 2-2.5% return premium (alpha) over a random selection of loans over all buckets, after factoring in effects from the simplifying assumption referenced earlier.

We will now move into our out-of-sample results. In order to validate the preliminary findings above, we trained the model on the entirety of the 2007-2011 dataset and tested on a newly cleaned 2012-2013 dataset. In order to ensure the time safety of our model, we never allowed any data from the latter set to make its way into the model as training observations, nor did we retroactively modify the model after observing the results on the out-of-sample data. As before, we note the general increase in differential as

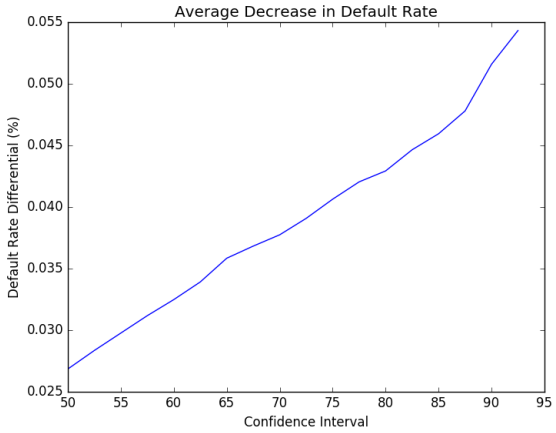

Figure 6: In-sample results averaged across loan buckets

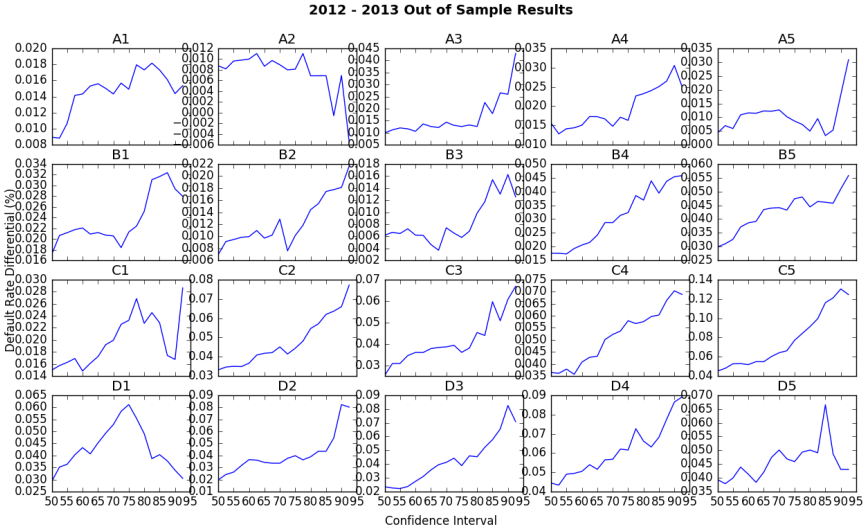

Figure 7: Out-of-sample results in loan buckets A-D

confidence increases. These increases are particularly notable and consistent in certain loan buckets, which supports the hypothesis that the LDA assumptions hold to a greater degree among some buckets as opposed to others. Additionally, some of the fluctuation, especially at the higher confidence intervals, is undoubtedly due to the aforementioned variance associated

with small loan sample selection. We note positive differentials across all of the buckets. Again, this chart reduces some of the variation contained in

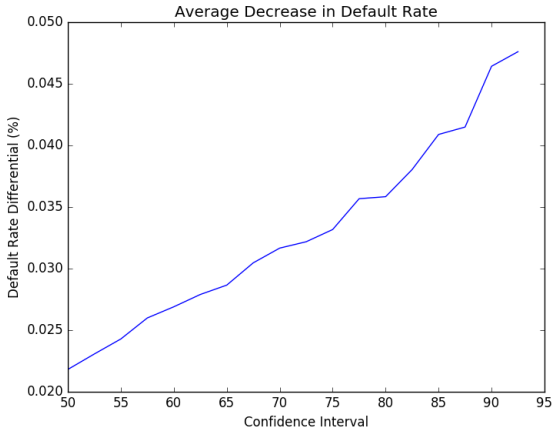

Figure 8: Out-of-sample results averaged across loan buckets

the prior figure. We note a strong linear correlation between our model's confidence and the observed differential that bears a high degree of similarity to that of the analogous in-sample results. Finally we have a table that gives precise values for the proportions of perfect loans selected by our model compared with the median proportion of perfect loans by C sub-bucket.

Results @70% Confidence					
	C1	C2	C3	C4	C5
Perfect loans (model)	0.848681542	0.863111111	0.81804203	0.817066667	0.817001181
Perfect loans (random)	0.828748783	0.818024263	0.779452476	0.764837626	0.752878654
Differential	0.019932759	0.045086848	0.038589554	0.052229041	0.064122527

Results @80% Confidence					
	C1	C2	C3	C4	C5
Perfect loans (model)	0.851491175	0.872666667	0.824750192	0.8224	0.844109832
Perfect loans (random)	0.828748783	0.818024263	0.779452476	0.764837626	0.752878654
Differential	0.022742392	0.054642403	0.045297716	0.057562374	0.091231178

Results @90% Confidence					
	C1	C2	C3	C4	C5
Perfect loans (model)	0.845498783	0.884	0.840245776	0.8352	0.883185841
Perfect loans (random)	0.828748783	0.818024263	0.779452476	0.764837626	0.752878654
Differential	0.016750001	0.065975737	0.0607933	0.070362374	0.130307187

Table 2: Selected results from C bucket loans

5 Generating a Price of a Lending Club Note

In this section, we will explore how to go from a probability output from a statistical learning algorithm to a dollar price on a peer-to-peer note. The first property we would like in a pricing model is that it is independent of the statistical model that generated the input. Every model in use has the excellent property that it either directly or indirectly outputs a probability that the loan will default at some point in its term. We will explore this over several models that we tried.

5.1 The Probability Output of Statistical Models

Logistic Regression: Were we to fit a logistic regression to classify the data, we would get a probability for free! This is given by the sigmoid likelihood hypothesis over a loan x and a fitted θ,

$$P(\text{default of } x) = h_\theta(x) = \frac{1}{1 + e^{\theta^T x}}.$$

Linear Discriminant Analysis: In linear discriminant analysis, we assume the data fits a Gaussian with the same variance but different means for

each class label. In this case, for a loan x and an indicator of default y, this gives the formula

$$p(x|y;\mu_y,\Sigma) = \mathcal{N}(\mu_y,\Sigma).$$

By applying Bayes' rule we can find the posterior probability $P(y|x)$ for the pricing model.

Support Vector Machines: The Support Vector Machine is a more difficult problem: We can solve the problem of assigning a probability to the output of the SVM model with Platt Scaling [8]. Without going into excessive detail, this method involves fitting a maximum-likelihood logistic regression to the results of the SVM model. In any case, this is a standard method and allows us to use SVM outputs.

Ensemble Methods/AdaBoost According to Niculescu et. al, we can again use Platt Scaling to achieve a good estimate of posterior probabilities from decision trees, random forests, and Adaboost [9].

5.2 The Pricing Model

In any case, it is safe for our pricing model to assume that there will be an output probability. This is the only output we are guaranteed to get. As a first attempt, we will construct a pricing formula for a note. Moving from there, we will add a correction for an obvious flaw in the model.

We first notice that is a loan defaults, this doesn't preclude the borrower from having similar problematic events occur in the rest of the loan term. So we must adjust our figure for the probability of default to account for this discrepancy.

Drawing from some of the work of Anderson [1], We take the implied or output probability of a given ML model P_{model}. Then we can model a default as a single event drawn from a Poisson distribution of which there could be many more in a given loan term. Let λ be the implied Poisson average of the number of defaults in a given loan period. Then we have that

$$P_{model} = \sum_{k=1}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!}. \tag{1}$$

We can then solve numerically for λ. In the model, this is implemented using a modified form of Newton's method.

After we have calculated the parameter for the Poisson model, we use a modified DCF analysis to give a total value of the bond. The modifications essentially account for the Poisson expectiation of default by weighting each payment by the likelihood of at least one default before they are paid off. If our default model were perfect, any LendingClub note worth over $25 would be a sound investment.

Let t be the term of a given loan in months. Then the implied monthly Poisson average $\gamma = \frac{\lambda}{t}$. Then we can break our probability of default into the Poisson expectation of no default in a month, $e^{-\gamma}$, the monthly payout of a loan c, and the discount rate, d, to give the following formula for the price V of a loan.

$$V = \sum_{m=1}^{t} \frac{e^{-m\gamma} c}{d^m}. \tag{2}$$

This pricing model has shown to be very numerically stable, but there are 2 serious objections to it.

1. The Poission assumption of default is not sound. This is an entirely valid criticism, but the data don't seem to support the claim that the loans default disproportionately at any particular part of the term. If we were to use a different probabilty distiribution, we would need richer input to make a maximum-likelihood estimate of the parameters, and we do not have any temporal data to make that feasible.

2. The model is entirely independent of possible upcoming changes in market conditions and the economy as a whole, so loans and their prices are totally vulnerable to a widespread downturn in the economy. This is also a valid complaint. It's also the *big* question, so a satisfying answer would probably be a much bigger deal than picking good consumer loans. That being said, there is one modification we can make, also from Anderson [1]. There is for some reason a large empirical effect of the 90-day health of loans of a particular vintage and the long-term prospects of the vintage. There is not a lot of work that has been done in this area, and we don't have the temporal data infrastructure to handle it, but we have here a modified formula that could be calibrated for a more sophisticated price.

   Let K be the 90-day non-delinquncy rate of loans in the past 6 months in the same debt grade and $\alpha > 1$ some constant that needs to be fit.

Then we can replace Equation 1 with the following

$$K^{-\alpha} P_{model} = \sum_{k=1}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \tag{3}$$

This is the most accurate pricing model we were able to create.

5.3 Sharpe Ratio

It remains to try and calculate a Sharpe ratio for the loan data. Since we didn't have the type of temporal back-testing infrastructure we would like, we were unable to empirically determine our Sharpe ratio. Instead, we derived an estimate based on the work of Bams empirically determining loan correlation of small retail loans in a variety of markets to be about 0.3[2]. Clearly, this is a vast oversimplification, and we are throwing away a massive amount of information. However, since this is for a retroactive performance measure, this is not a top priority.

For purchased loans x with variance in earnings σ_x and value V_x from Equation 2 and cost C, let $\sigma = \sum_x \sigma_x$, $R_a = \sum_x V_x$, and $R_b = \sum_x C$. Then the Sharpe S can be calculated in the usual way by

$$S = \frac{E[R_a - R_b]}{\sigma_a}.$$

These formula are all sums over the number of loans purchased, so we actually used online averages for ease of computation. Under the buying strategy discussed, our Sharpe was 3.41.

6 Risk Management

Like any investment strategy, there are always associated risks. Therefore, we took a few measures to minimize our exposure to some of these risks. We were initially quite worried about the accuracy of our model over multiple years since we built our models based on the 2007-2011 dataset provided by Lending Club. Was this snapshot in time indicative of ever-changing conditions in the market? For example, are people in 2009 just as likely to pay off loans as they are in 2012? Fortunately, upon outside testing of our model on the 2012 and 2013 datasets (which were not used to build the model) we

still achieved similar results.

Vik Chawla, a lead research associate at Echelon Capital Management, explained that the primary concern with changing market conditions is that this industry has not been around during a market cycle. We have limited data about the sustainability of the model in the event of a major downturn in the market. As a result, we are focusing primarily in 36-month loans, rather than the 60-month loans. The shorter loans decrease the probability that they will overlap with a downturn in the market minimizing our exposure to this uncertainty as much as possible.

The other major risk, which is much more difficult to control, is if Lending Club were to go out of business. Mr. Chawla explained that while this risk is most certainly not negligible, it is a risk that is carried with any investment. Additionally, Lending Club has some safeguards in place to ensure loan payment in the event that Lending Club goes out of business. Lending Club has executed a backup and successor servicing agreement with Portfolio Financial Servicing Company ("PFSC"). Under this agreement, PFSC will service borrower loans if Lending Club is unable to.

7 Execution

Execution became a key question for our group - unlike other groups, who traded stocks or securities, we had no pre-existing infrastructure (such as Quantopian or Thesys) that would automatically implement our model. In order to automate our model process, we wrote a Python script that brought together our various models and automatically purchased loans that met our predetermined criteria. The following graphic illustrates our execution path, which we will walk through step by step.

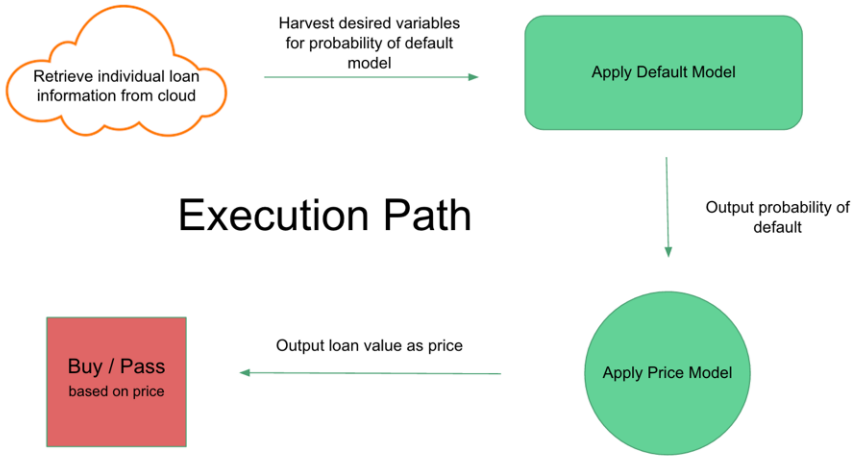

Figure 9: Execution flowchart

Lending Club offers a RESTful API, a feature that ultimately became critical to our execution progression. Using the Requests Python module, we were able to access the most recent batch of loans through the following line: The rq represents the Requests package, which allows us to make this

```
loans = rq.get('https://api.lendingclub.com/api/investor/v1/loans/listing', headers={'Authorization': 'Y8tVGnK3UGwA+jqURY/YYlrpEzc='})
```

one-line HTTP call. The URL represents the specific path request that we are making (in this case, loan listings) and the Authorization is a unique code attached to our Lending Club account. This way, we don't need to log in with our usual username and password (i.e. the script does not need to worry about login time-out). This request returns a JSON file representing the most recent batch of loans. Our first major step is to parse this data. We get the relevant features of each loan (see code appendix for details), and feed this data into a separate module. This module represents our primary model, which takes in the various fields and applies the corresponding LDA (as discussed earlier). This function returns our predicted probability of default (again, see appendix for details).

We take this probability and feed it into our price function, which implements the algorithm described earlier in this paper and returns a price (per $25 invested). This price represents our calculated present value of the loan

(it even discounts inflation, which for our purposes is set at 2% a year). For example, if the price returned is $27.5 and we purchase $25 of that loan, we would add a book value of $27.50 to our portfolio. If the price returned is $24.50, we would add $24.50 to our portfolio book value. We then take that price and either purchase or do not purchase the loan based on whatever threshold we input. The following request executes the purchase:

```
purchase = rq.post('https://api.lendingclub.com/api/investor/v1/accounts/83008979/orders',
 headers={'Authorization': 'Y8tVGnK3UGwA+jqURY/YYlrpEzc='}, json={"aid":83008979, "orders":[{"loanID": id_num, "requestedAmount": 25.0}]})
```

This is the only other occasion that we directly interact with the Lending Club servers during the entire process. The URL and Authorization are identical with the earlier call, and the JSON payload conveys the loan ID of the loan to be purchased, as well as the amount that we plan to purchase. We can run this script in the background permanently, constantly checking for the newest batch of loans and acting accordingly.

Moving forwards, we have several areas of opportunity to make our execution more effective. From talking to Vik, we learned of two specific focus areas. First, we can colocate our servers in Las Vegas, where the Lending Club servers are. Other firms in the space have spent individually more than $200,000 on this endeavour (Vik), but if we really wanted to implement our model on a large scale, it can be worth it to get a first mover advantage on loans coming out. Secondly, we can work on a predictive model to assess when the most valuable loans are likely to surface. Lending Club strictly enforces a limit of one API interaction (listing request, purchase, etc) per second, so adding a level of finesse to our searches could offer a huge advantage. For example, rather than grabbing a random loan per second to evaluate, we could tune our search to identify different loans at different times, based on the probability of their appearance.

However, while we are still at this stage of operations, the competition factor should not be as critical. Lending Club offers a different set of loans to institutional and large-scale investors than to "amateur" investors. These different categories are identical in loan composition and are determined randomly, and with a split of 80% to 20% in favor of the amateur section. We fall firmly in this amateur category, and so long as we stay there, latency and time prediction will not make or break our strategy.

Reflection and Acknowledgements

Having touched on a number of areas for future work throughout the paper, we don't believe it necessary to restate those avenues here. Please see the above sections on modeling, pricing, and execution for specific prospective future directions pertaining to this project.

As regards acknowledgements, we would like to sincerely thank Dr. Lisa Borland and Enzo Busseti for their help and guidance throughout the course, especially in facilitating connections with industry contacts. Jeff Hilton and Vik Chawla were also tremendous resources and were more than generous in offering their time and expertise in advising us. Finally, we would also like to offer our heartfelt congratulations to all of our fellow students in 448 this quarter - all of the projects were highly informative and well done.

References

[1] Scott Anderson and Janet Jowzik. Building a credit model using gse loan-level data. *The Journal of Structured Finance*, 20:19–36, 2014.

[2] Wilhelmus Fransiscus Maria Bams, Magdalena Pisa, Christian CP Wolff, et al. *Modeling default correlation in a US retail loan portfolio.*

[3] Chris Barth. Looking For 10% Yields? Go Online For Peer To Peer Lending.

[4] Contributor. Peer To Peer Lending Crosses $1 Billion In Loans Issued.

[5] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182, March 2003.

[6] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: with Applications in R.* Springer, New York, 1st ed. 2013, corr. 5th printing 2015 edition edition, August 2013.

[7] Ron Kohavi and George H. John. Relevance wrappers for feature subset selection. *Artificial Intelligence*, 97(1):273 – 324, 1997.

[8] John C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *ADVANCES IN LARGE MARGIN CLASSIFIERS*, pages 61–74. MIT Press, 1999.

[9] Peter Renton. Obtaining calibrated probabilities from boosting. *CoRR*, abs/1207.1403, 2012.

Appendix: Code

Main execution script:

```
import requests as rq
import time
from invest import decide
import warnings
from price import get_price
import sys
warnings.filterwarnings("ignore")

def getSub(subgrade):
  if subgrade == 'A1':
    return 0, .0532
  if subgrade == 'A2':
    return 1, .0649
  if subgrade == 'A3':
    return 2, .0697
  if subgrade == 'A4':
    return 3, .0739
  if subgrade == 'A5':
    return 4, .0789
  if subgrade == 'B1':
    return 5, .0839
  if subgrade == 'B2':
    return 6, .0916
  if subgrade == 'B3':
    return 7, .0975
  if subgrade == 'B4':
    return 8, .1075
  if subgrade == 'B5':
    return 9, .1147
  if subgrade == 'C1':
    return 10, .1199
  if subgrade == 'C2':
    return 11, .1299
  if subgrade == 'C3':
```

```
35      return 12, .1367
36    if subgrade == 'C4':
37      return 13, .1446
38    if subgrade == 'C5':
39      return 14, .1531
40    if subgrade == 'D1':
41      return 15, .1629
42    if subgrade == 'D2':
43      return 16, .1727
44    if subgrade == 'D3':
45      return 17, .1825
46    if subgrade == 'D4':
47      return 18, .1899
48    else:
49      return 19, .1999
50
51
52  def processLoan(results, index):
53    subgrade = results[index]['subGrade']
54    if subgrade == 'E1' or subgrade == 'E2' or
         subgrade == 'E3' or subgrade == 'E4' or
         subgrade == 'E5' or subgrade == 'F5' or
         subgrade == 'F4' or subgrade == 'F3' or
         subgrade == 'F2' or subgrade == 'F1' or
         subgrade == 'G1' or subgrade == 'G2' or
         subgrade == 'G3':
55      return
56    subgrade_input, rate = getSub(subgrade)
57    term = results[index]['term']
58    if term == 36:
59      term_input = 0
60    else:
61      term_input = 1
62      return
63    id_num = results[index]['id']
64    loan_amt = results[index]['loanAmount']
65    emp_length = results[index]['empLength']
66    if emp_length == None:
```

```
67      emp_length = 0
68    home = results[index]['homeOwnership']
69    if home == 'MORTGAGE':
70      home_input = 1
71    elif home == 'OWN':
72      home_input = 2
73    else:
74      home_input = 0
75    dti = results[index]['dti']
76    inc = results[index]['annualInc']
77    verified = results[index]['isIncV']
78    if verified == 'VERIFIED':
79      ver_input = 2
80    elif verified == 'SOURCE_VERIFIED':
81      ver_input = 1
82    else:
83      ver_input = 0
84    inq = results[index]['inqLast6Mths']
85    openAccs = results[index]['openAcc']
86    revoBal = results[index]['revolBal']
87    revoUt = results[index]['revolUtil']
88    totalAcc = results[index]['totalAcc']
89    print 'subgrade: ' + subgrade, '\nterm:', term, '\
        nloan ID:', id_num, '\nloan amount:', loan_amt,
         '\nemployment length:', emp_length, '\nhome
        ownership:', home, '\nincome:', inc, '\nDTI:',
        dti, '\nincome verified?', verified, '\nnumber
        of inquiries:', inq, '\nopen accounts:',
        openAccs, '\nrevolving balance:', revoBal, '\
        nrevolving utility:', revoUt, '\ntotal accounts
        :', totalAcc, '\n'
90    loan = [term_input, loan_amt, emp_length/12.0,
        home_input, inc, ver_input, dti, inq, openAccs,
         revoBal, revoUt/100.0, totalAcc]
91    inves, percent = decide(loan, subgrade_input, 80)
92    price = round(get_price(1 - percent, rate, term,
        1.04), 2)
93    if price > 27:
```

```
94      print 'Our␣model␣tells␣us␣that␣we␣SHOULD␣buy␣
           this␣loan,␣as␣it␣has␣a', 100 - round(percent
           * 100, 2), "percent␣chance␣of␣default,␣and␣an
           ␣estimted␣value␣of␣$", price, 'per␣$25␣
           purchased,␣given␣an␣interest␣rate␣of', rate
95    else:
96      print 'Our␣model␣tells␣us␣that␣we␣SHOULD␣NOT␣buy
           ␣this␣loan,␣as␣it␣has␣a', 100 - round(percent
            * 100, 2), "percent␣chance␣of␣default,␣and␣
           an␣estimted␣value␣of␣$", price, 'per␣$25␣
           purchased,␣given␣an␣interest␣rate␣of', rate
97    print '\n'
98    buy = input("Should␣we␣purchase?␣1␣to␣purchase,␣0␣
         to␣pass,␣2␣to␣end:␣")
99    print '\n'
100   if buy == 2:
101     sys.exit(0)
102   if buy == 1:
103     po = rq.post('https://api.lendingclub.com/api/
           investor/v1/accounts/83008979/orders',
           headers={'Authorization': 'Y8tVGnK3UGwA+jqURY
           /YYlrpEzc='}, json={"aid":83008979, "orders"
           :[{"loanID": id_num, "requestedAmount":
           25.0}]})
104     print 'loan', id_num, "was␣not␣successfully␣
           purchased,", 'INSUFFICIENT_CASH\n'
105   time.sleep(1)
106
107 print '\nWelcome␣to␣the␣Team␣4␣automated␣trading␣
       script␣\n'
108 time.sleep(1)
109 print 'Created␣by␣Andy␣Feis,␣Viraj␣Mehta,␣Scott␣
       Morris,␣John␣Solitario,␣and␣Cameron␣Van␣Der␣Graaf
       ␣\n'
110 time.sleep(1)
111 print 'Retrieving␣most␣recent␣loans...\n'
112 time.sleep(1)
113
```

```
114 loans = rq.get('https://api.lendingclub.com/api/
      investor/v1/loans/listing', headers={'
      Authorization': 'Y8tVGnK3UGwA+jqURY/YYlrpEzc='})
115
116 json_loans = loans.json()
117 results = json_loans['loans']
118 for index in range(len(results)):
119   processLoan(results, index)
```

Investment Decision module:

```
 1 import numpy as np
 2 import pandas as pd
 3 from sklearn.lda import LDA
 4 import pickle
 5 import warnings
 6
 7 warnings.filterwarnings("ignore")
 8
 9 def decide(loan, loan_grade, confidence):
10   pred_arr = []
11   res_arr = []
12   for i in range (0,20):
13     pred_arr.append(pickle.load(open("/Users/Scott/
         Desktop/448execution/lda_" + str(i) + ".p", "
         rb" )))
14     res_arr.append(pickle.load(open("/Users/Scott/
         Desktop/448execution/res_" + str(i) + ".p", "
         rb" )))
15   q = np.percentile(res_arr[loan_grade], confidence)
16   return pred_arr[loan_grade].predict_proba(loan)
        [0,1] > q, pred_arr[loan_grade].predict_proba(
        loan)[0,1]
```

Pricing module:

```
1 import math
2
3
4 def get_lambda(p):
```

```
5   lambda_up = 1.0
6   lambda_down = 0
7   for i in range(1,200):
8    l = (lambda_up + lambda_down) / 2.0
9    p_0 = 0
10   for k in range(1,150):
11    p_0 += math.exp(float(k) * math.log(l) - l - math
      .log(math.gamma(float(k) + 1)))
12   if p_0 > p:
13    lambda_up = l
14   else:
15    lambda_down = l
16
17  return (lambda_up + lambda_down) / 2
18
19 def get_price(p, interest_rate, months, discount):
20  """We're assuming here that the coupon value is $25
      , but that can be easily generalized.
21  Enter the interest rate as a decimal 0 < interest
      rate < 1. This is assuming that this is an an APR
      ."""
22  monthly_interest = interest_rate /12
23  coupon = 25 * (monthly_interest / (1 - (1 +
      monthly_interest)**(-1 * months)))
24  l = get_lambda(p)
25  gamma = l / float(months)
26  monthly_discount = ((discount - 1) / 12) + 1
27  # print gamma
28  V = 0
29  for m in range(1, months):
30   V += (math.exp(-1 * m * gamma) * coupon) / (
      monthly_discount ** m)
31  return V
32
33 # def main():
34 #  print get_price(0.01, 0.05, 36, 1.01)
35
36 # if __name__ == '__main__':
```

```
37 #  main()
```

Model creation script 1:

```
1  # In-sample analysis
2  import matplotlib.pyplot as plt
3  from matplotlib.ticker import MaxNLocator
4  import numpy as np
5  import pandas as pd
6  from sklearn import linear_model
7  from sklearn import cross_validation
8  from sklearn import ensemble
9  from sklearn import svm
10 from sklearn.lda import LDA
11 from scipy import stats
12 from sklearn import grid_search
13 import statsmodels.api as sm
14 import pickle
15
16
17 pred_arr = []
18
19 base = "/Users/Cameron/Documents/MSE448/Data/Test
      /2007_2011_week9_"
20 suffix = ".csv"
21 letters = ["a", "b", "c", "d"]
22 numbers = ["1", "2", "3", "4", "5"]
23 l = [base + x + y + suffix for x in letters for y in
       numbers]
24
25 def_rates = {l[0]:.0603, l[1]:.0662, l[2]:.0751, l
      [3]:.079, l[4]:.089,
26        l[5]:.0991, l[6]:.1065, l[7]:.1171, l
             [8]:.1242, l[9]:.1269,
27        l[10]:.1349, l[11]:.1427, l[12]:.1465, l
             [13]:.1527, l[14]:.1596,
28        l[15]:.1629, l[16]:.1677, l[17]:.1727, l
             [18]:.1758, l[19]:.1825}
29
```

```
30 confidence_ints = np.arange(50,95,2.5)
31
32 results_by_bucket = {}
33
34 runs = 20
35
36 for n in range(0,runs):
37   c_acc = []
38   for z in confidence_ints:
39     acc = []
40     select = []
41     for f in l:
42       # print f
43       arr = pd.read_csv(f).dropna()
44       arr_np = arr.values
45
46       X = arr_np[:,2:]
47       Y = np.ravel(np.asarray(arr_np[:,:1], dtype='d
           '))
48       default_rate = np.mean(Y)
49
50       X_train, X_test, y_train, y_test =
           cross_validation.train_test_split(
51         X, Y, test_size=0.33, random_state=n)
52
53       lda = LDA()
54       lda.fit(X_train, y_train)
55       res = lda.predict_proba(X_test)[0:,1]
56       q = np.percentile(res,z)
57
58       #j is counter, c is number of non-default
           loans, b is num total invested loans
59       j = 0
60       c = 0
61       b = 0
62       for i in res:
63         if i > q:
64           if y_test[j] == 1:
```

```
65              c += 1
66            b += 1
67          j += 1
68
69        acc.append(def_rates[f] - (1 - (float(c)/b)))
70
71        if n == 0:
72          results_by_bucket[f] = {n : [def_rates[f] -
               (1 - (float(c)/b))]}
73        else:
74          results_by_bucket[f][n] = results_by_bucket[
               f][n].append(def_rates[f] - (1 - (float(c
               )/b)))
75
76        # print (def_rates[f] - (1 - ((float(c)/b) -
             np.mean(y_test)))) * 100
77        pred_arr.append(lda)
78      c_acc.append(np.mean(acc))
79    plt.plot(confidence_ints, c_acc)
80      print "cumulative accuracy = " + str(np.mean(acc
          )) + "q = " + str(z)
81
82  letters = ["a", "b", "c", "d"]
83  numbers = ["1", "2", "3", "4", "5"]
84  lablels = [x + y for x in letters for y in numbers]
85
86  fig, axarr = plt.subplots(4,5,figsize=(15,8))
87  fig.suptitle('2007 - 2011 Test Results', fontsize
       =14, fontweight='bold')
88  fig.text(0.5, 0.04, 'Confidence Interval', ha='
       center')
89  fig.text(0.08, 0.5, 'Default Rate Differential (%)',
        va='center', rotation='vertical')
90
91  for ax, f in enumerate(l):
92
93    arr = pd.read_csv(f).dropna()
94    arr_np = arr.values
```

```
95    X = arr_np[:,2:]
96    Y = np.ravel(np.asarray(arr_np[:,:1], dtype='d'))
97    default_rate = np.mean(Y)
98
99    results_by_bucket[f] = np.ndarray(shape=(runs,len(
          confidence_ints)))
100
101   for n in range(0,runs):
102     for count, z in enumerate(confidence_ints):
103
104       X_train, X_test, y_train, y_test =
              cross_validation.train_test_split(
105           X, Y, test_size=0.33, random_state=n)
106
107       lda = LDA()
108       lda.fit(X_train, y_train)
109       res = lda.predict_proba(X_test)[0:,1]
110       q = np.percentile(res,z)
111
112       #j is counter, c is number of non-default
              loans, b is num total invested loans
113       j = 0
114       c = 0
115       b = 0
116       for i in res:
117         if i > q:
118           if y_test[j] == 1:
119             c += 1
120           b += 1
121         j += 1
122
123       results_by_bucket[f][n,count] = float(c)/b -
              np.mean(Y)
124
125     axarr[ax/5, ax%5].plot(confidence_ints,
            results_by_bucket[f][n,:])
126     axarr[ax/5, ax%5].set_title(lablels[ax])
127     # plt.subplots_adjust(wspace=0.4, hspace=0.4)
```

```
128
129     # axarr[c/5, c%5].set_title('Axis [0,0]')
130 plt.setp([a.get_xticklabels() for x in range(0,3)
       for a in axarr[x, :]], visible=False)
131 plt.savefig("2007_2011_individual.png")
132 plt.close()
133
134 cumulative_accuracy = []
135 for n in range(0,len(confidence_ints)):
136   acc = []
137   for bucket in results_by_bucket:
138     acc.append(np.mean(results_by_bucket[bucket],
          axis=0)[n])
139   cumulative_accuracy.append(np.mean(acc))
140 figure = plt.plot(confidence_ints,
       cumulative_accuracy)
141 plt.xlabel('Confidence␣Interval')
142 plt.ylabel('Default␣Rate␣Differential␣(%)')
143 plt.title('Average␣Decrease␣in␣Default␣Rate')
144 plt.savefig('2007_2011_avg.png')
145
146 for x in l:
147   plt.subplot()
148   for i in range(0,runs):
149     print results_by_bucket[x][i]
150     plt.plot(confidence_ints, results_by_bucket[x][i
          ])
151     plt.yscale('linear')
152     plt.grid(True)
153
154 plt.show()
155
156
157 for i in range(0,len(l)):
158   print "in␣loan␣grade␣" + str(l[i]) + "␣accuracy␣=␣
        " + str(acc[i]) + "␣and␣proportion␣selected␣=␣"
         + str(select[i])
159   print ""
```

Model creation script 2:

```
# Out-of-sample analysis and model export
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
import numpy as np
import pandas as pd
from sklearn import linear_model
from sklearn import cross_validation
from sklearn import ensemble
from sklearn import svm
from sklearn.lda import LDA
from scipy import stats
from sklearn import grid_search
import statsmodels.api as sm
import pickle

base = "/Users/Cameron/Documents/MSE448/Data/Test/2012/loan_info_012-13"
suffix = ".csv"
letters = ["A", "B", "C", "D"]
numbers = ["1", "2", "3", "4", "5"]
test_set = [base + x + y + suffix for x in letters for y in numbers]

base = "/Users/Cameron/Documents/MSE448/Data/Test/2007_2011_week9_"
suffix = ".csv"
letters = ["a", "b", "c", "d"]
numbers = ["1", "2", "3", "4", "5"]
train_set = [base + x + y + suffix for x in letters for y in numbers]

def_rates = {test_set[0]:.0603, test_set[1]:.0662, test_set[2]:.0751, test_set[3]:.079, test_set[4]:.089,
    test_set[5]:.0991, test_set[6]:.1065, test_set[7]:.1171, test_set[8]:.1242, test_set
```

```
            [9]:.1269,
30          test_set[10]:.1349, test_set[11]:.1427,
            test_set[12]:.1465, test_set[13]:.1527,
            test_set[14]:.1596,
31          test_set[15]:.1629, test_set[16]:.1677,
            test_set[17]:.1727, test_set[18]:.1758,
            test_set[19]:.1825}
32
33  confidence_ints = np.arange(50,95,2.5)
34
35  results_by_bucket = {}
36
37  letters = ["A", "B", "C", "D"]
38  numbers = ["1", "2", "3", "4", "5"]
39  lablels = [x + y for x in letters for y in numbers]
40
41  fig, axarr = plt.subplots(4,5,figsize=(15,8))
42  fig.suptitle('2012 - 2013 Out of Sample Results',
      fontsize=14, fontweight='bold')
43  fig.text(0.5, 0.04, 'Confidence Interval', ha='
      center')
44  fig.text(0.08, 0.5, 'Default Rate Differential (%)',
       va='center', rotation='vertical')
45
46  master_y_test = []
47  master_y_res = []
48
49  for ax, f in enumerate(test_set):
50
51    train = pd.read_csv(train_set[ax]).dropna()
52    train_np = train.values
53    X_train = train_np[:,2:]
54    y_train = np.ravel(np.asarray(train_np[:,:1],
        dtype='d'))
55
56    test = pd.read_csv(f).dropna().replace(to_replace=
        'NONE', value=0).replace(to_replace='OTHER',
        value=0)
```

```
57    test_np = test.values
58    X_test = test_np[:,3:]
59    y_test = np.ravel(np.asarray(test_np[:,1], dtype='
         d'))
60    master_y_test.extend(y_test)
61
62    results_by_bucket[f] = np.empty(len(
         confidence_ints))
63
64    lda = LDA()
65    lda.fit(X_train, y_train)
66    pickle.dump(lda, open("lda_" + str(ax) + ".p", "w"
         ))
67    res = lda.predict_proba(X_test)[0:,1]
68    pickle.dump(res, open("res_" + str(ax) + ".p", "w"
         ))
69    master_y_res.extend(res)
70
71    for count, z in enumerate(confidence_ints):
72      q = np.percentile(res,z)
73
74      #j is counter, c is number of non-default loans,
           b is num total invested loans
75      j = 0
76      c = 0
77      b = 0
78      for i in res:
79        if i > q:
80          if y_test[j] == 1:
81            c += 1
82          b += 1
83        j += 1
84
85      print f
86      print z
87      print float(c)/b
88      print np.mean(y_test)
89      print float(c)/b - np.mean(y_test)
```

```
90
91      results_by_bucket[f][count] = float(c)/b - np.
           mean(y_test)
92
93    axarr[ax/5, ax%5].plot(confidence_ints,
         results_by_bucket[f])
94    axarr[ax/5, ax%5].set_title(lablels[ax])
95      # plt.subplots_adjust(wspace=0.4, hspace=0.4)
96
97      # axarr[c/5, c%5].set_title('Axis [0,0]')
98  plt.setp([a.get_xticklabels() for x in range(0,3)
       for a in axarr[x, :]], visible=False)
99  plt.show()
100 plt.savefig('2012_2013_individual.png')
101 plt.close()
102
103 cumulative_accuracy = []
104 for n in range(0,len(confidence_ints)):
105   acc = []
106   for bucket in results_by_bucket:
107     acc.append(results_by_bucket[bucket][n])
108   cumulative_accuracy.append(np.mean(acc))
109 figure = plt.plot(confidence_ints,
       cumulative_accuracy)
110 plt.xlabel('Confidence␣Interval')
111 plt.ylabel('Default␣Rate␣Differential␣(%)')
112 plt.title('Average␣Decrease␣in␣Default␣Rate')
113 # plt.savefig('2012_2013_avg.png')
114 plt.close()
115
116
117 total_loans_invested = 0
118 non_default = 0
119 const = np.percentile(master_y_res,90)
120 print const
121 for i in range(0,len(master_y_test)):
122   if np.random.randint(10) == 5 and master_y_res[i]
        > const:
```

```
123     total_loans_invested += 1
124     if master_y_test[i] == 1:
125       non_default += 1
126
127 print non_default/total_loans_invested
128 print np.mean(master_y_test)
```