Predicting Probability of Loan Default
Stanford University, CS229 Project report

Jitendra Nath Pandey, Maheshwaran Srinivasan

12/15/2011

Abstract: Extending credit to individuals is necessary for markets and societies to function smoothly. Estimating the probability that an individual would default on his/her loan, is useful for banks to decide whether to sanction a loan to the individual and is also useful for borrowers to make better financial decisions. In this project, we used supervised learning to estimate the probability of loan default for individuals. Training and test data were drawn from a competition on Kaggle [1]. The metric used to judge the efficiency of a solution was the AUC (area under the ROC Curve) calculated on probabilities of default for the test data. We tried Logistic regression, Probit Regression, Support Vector Machines (SVM) using an RBF (Radial Basis Function) Kernel, Classification trees and finally Random Forests. We obtained the best result (AUC score of 0.867125) with Random Forests Regression, using over-sampling to mitigate class 1/0 imbalance and using carefully chosen coded values for missing values.

1. Introduction: Our goal was to estimate the probability that a person applying for a loan will experience 90 days past due on loan payments or delinquency or an even more serious financial crisis. The training data, drawn from the competition ***'Give me some Credit'*** on Kaggle [1], contained records on 150,000 people. Each training record had information on 10 input features and a binary (0/1) dependent variable. The test data had records for 101,503 people containing information on the above 10 features and we had to predict probability of loan default for these people based on the input features. The effectiveness of the prediction was judged by calculating the AUC (area under the ROC curve) from the probabilities of default for the test records. The AUC calculated on all the 101,503 test records (which shall be referred to as AUC_FULL) was made available to us only at the end of the competition, just two days before the project deadline. An AUC calculated on 30% of the test records (which shall be referred to as AUC_PART) was available to us throughout the competition. Our decision making on algorithm selection was hence guided somewhat by AUC_PART in addition to AUC scores computed on held out training data. Luckily, our AUC_PART scores correlated very well with our AUC_FULL scores.
The AUC metric for a 2-class problem is calculated as: Sort the estimated probabilities in ascending order

$$AUC = \frac{(S_1 - n_1 * \frac{n_1+1}{2})}{n_0 * n_1}$$ where, S_1 is the sum of the ranks of class-1 records and n_0 (n_1) is the number of items which are actually class-0(1)

2. Data: The training and test data suffered from the following problems:

- Missing values for 2 features. As much as 21% of the records have missing features.
- Coded values for 3 other features. Only 0.2% of the records have coded values.
- Most of the input features had outliers. For instance, Ruul, a feature with high predictive power, normally in the range [0,1], sometimes had anomalously high values (50,000+)
- Class imbalance in training data i.e. number of records with class-1 labels was only 6% of the total records.

3. Algorithms and Results

3.1 Logistic and Probit Regression: In Logistic regression, $P(Y = 1|X; \beta) = \frac{1}{1+e^{-\beta^T X}}$

In Probit regression, $P(Y = 1|X; \beta) = \Phi(\beta^T X)$ where Φ is CDF of N(0,1) (Standard normal)

3.1.1 Multiple Models: Since the parameter β could be quickly computed even when the training data had more than 100,000 records, to solve the problem of missing data in the test records, we developed 8 different models, each model using a different subset of input features. For instance one model used all 10 features and was employed when all features were present, another model used only input features 1..9 and was employed when feature10 was missing and so on. Thus depending on the features that were

"clean" in the test data record, we would use the appropriate model to compute the probability of a positive (class 1).

3.1.2 Feature Saturation and Under-sampling: We were able to get a dramatic leap in performance by saturating an important feature with high predictive power (Ruul) to a maximum value (Ruul_max) that was determined by 10-fold Cross validation in the test data and throwing away training records which had Ruul > Ruul_max. We obtained a further small increase in the AUC_PART score by under-sampling class0 records so that ratio of class0 records to class1 records was 3:1 (found by 10-fold cross validation).

Figure 1

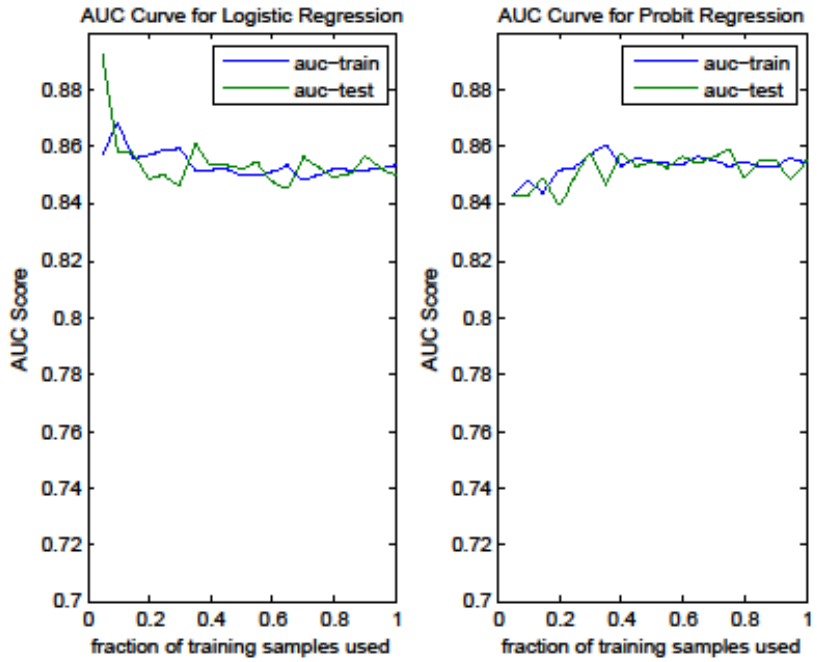

Table 1

Approach	**Logistic (AUC_PART)**	**Probit (AUC_PART)**
Multiple models	0.81116	0.81237
Multiple models with Feature saturation	0.84768	0.84918
Multiple models, Feature saturation, Data balancing	0.84919	0.8492

3.1.3 Diagnostics: As a diagnostic procedure, we plotted AUC curves (Training AUC and Test AUC with 5-fold Cross validation, versus size of training data) (See Fig1) for both logistic and Probit regression with multiple models, Ruul saturation and Data Balancing. From the AUC curves, we saw that training and test AUC were nearly the same for reasonable number of training samples. Hence we concluded that high variance was not a problem. Since we were targeting an AUC of above 0.86, we concluded that relatively high bias might be a problem which led us to explore Kernelized approaches.

3.2. Support Vector Machines

3.2.1: Choice of SVM, kernel and class probability estimation: We used the C-SVM with L1 regularization and RBF(radial basis function) Kernel. We focused on RBF kernel since, we seemed to be suffering from high bias and this Kernel maps input samples into an infinite dimensional space. Also RBF Kernel has relatively fewer model parameters to optimize, is numerically stable compared to high order polynomial kernels and some other kernels like Linear and Sigmoid kernel are special cases of the RBF Kernel under certain conditions [3].

We used the libsvm library [2] to implement the SVM with RBF Kernel for our problem. In order to generate posterior probabilities, libsvm uses an improved form of Platt's method [4] which consists of approximating the probability by a sigmoid function of the form:

$$P(y = 1|X) = \sigma(A\ f(X) + B) \quad \text{where,}$$

$$f(X) = w^T\phi(X) + b \quad \text{and } \sigma \text{ is the sigmoid function}$$

3.2.2: Methodology: We opted to train only three models due to the computational complexity of model training, again each model using a different subset of input features. For each test-record depending on the "clean" features available, we used the appropriate model. In the test data, we replaced coded values with their median, which was 0. We hoped that this would not have a significant negative impact since the number of such records was only 0.2% of the total. We used 5-fold cross validation with AUC as the metric to estimate optimal C and γ for each of the 3 models. We also saturated the Ruul parameter to a maximum value Ruul_max for all test records and threw out training records with Ruul > Ruul_max.
3.2.3: Under-sampling to handle imbalance of class-1 and class-0 records: Since cross validation is computationally expensive we used a smaller training data set of 20,000 "clean" records with a 3:1 ratio of class0 records to class1 records for C and γ estimation. Once optimal parameters were obtained, each model was retrained on a larger data set containing 32,000 clean records with again a 3:1 ratio of class0 to class1 records. We computed posterior probabilities with these 3 optimized models and obtained an AUC_PART score of 0.85013.
3.2.4: Class weights to handle imbalance of class-1 and class-0 records: Instead of using under-sampling, we also tried applying a more severe misclassification penalty for class-1 records. More specifically, libsvm gives the option of applying penalties C*w0 for errors in classifying class-0 records and C*w1 for errors in classifying class-1 records as opposed to a common penalty of C. We generated a training set of 20,000 records by randomly sampling from the "clean" training data. Then we set w0 to 1 and tried to find optimal values for C, γ and w1 by brute-force search over a well designed search space and using 5-fold cross-validation with AUC as metric. We found optimal parameters for each of the three models. Once optimal parameters had been obtained for the three different models, we re-trained the three models with these optimal parameters for C, γ and w1 over the entire "clean" training data containing around 118,000 records to predict class probabilities. We computed posterior probabilities and obtained an AUC_PART score of 0.84628.

3.3. Classification Trees (CART)

3.3.1: Description: In a classification tree, the input feature space is partitioned into a set of rectangles (regions) using recursive binary splits and then in each rectangle, a simple model like a constant is fitted[6]. Given a region R containing training records, to split the region into R1 and R2, all possible binary splits ($X_i < s$, $X_i >= s$) are examined and the (feature, value) pair (X_i^*, s^*) is selected that optimizes a particular criterion - minimize Gini index or Cross Entropy/Deviance or maximize "twoing" metric [5]. This process is repeated recursively for each child and we stop splitting if a node is pure (i.e. contains observations of only one class), or it has fewer than *MinParent* records or if any split imposed on this node would result in a child with less than *MinLeaf* records. For predicting class probabilities for a test record, the test record is "run" down the classification tree until it reaches a leaf node and then the class-1 probability is estimated as the fraction of class-1 elements in that leaf node.
3.3.2: Avoiding over-fitting: Deep trees are susceptible to over-fitting and there are two ways of controlling their depth and reducing generalization error. One method is to find an optimal value for the *MinLeaf* parameter (30 different values, varied logarithmically from 10 to 1000) using 10-fold cross-validation. The other method called pruning, is to grow a full-sized deep tree and then to reduce the tree depth by recursively merging leaf nodes. The optimal tree-depth to prune is decided using 10-fold cross-validation.
3.3.3: Handling missing values using surrogate variables: While building a classification tree, at each non-leaf node R, having chosen an optimal (primary) feature F and split point, we also choose secondary, tertiary… features and split points ranked according to the ability of the feature to mimic the split of the training data at node R according to the primary feature F. When a test record is "run" down the tree and hits node R, if it has primary feature F missing, then the best ranking non-missing feature is used for finding the child node to go to.
3.3.4: Results from using classification trees: We implemented classification trees using MATLAB Statistics toolbox. We experimented with three different node splitting criteria – Gini, Deviance and Twoing and both methods of controlling over-fitting – finding optimal value for *MinLeaf* and Pruning.

We were able to obtain best results with split criterion set to Twoing, using optimal value for MinLeaf and turning on the option to use Surrogate variables at each node. We obtained AUC_PART=0.85310.

3.4: Random Forests

3.4.1: Description: Random forest is an ensemble supervised learning method, which builds a large collection (500-2500) of de-correlated trees and then averages them [6]. Each tree in the ensemble is constructed by taking a boot-strap sample of size N_b from the training set and growing a classification or regression tree with that sample. While growing the tree, at each node, *m* out of the *P* input features are randomly selected to determine best feature for splitting at that node. Reducing *m*, reduces the correlation between the different trees but it also reduces the "strength" of each individual tree [7]. While predicting, a test record is "run" down each tree and the results over all trees in the ensemble are averaged for regression and a majority vote taken for classification. Usually the mean square error is used as the splitting criterion while building regression trees and the Gini index is used as the splitting criterion while building classification trees.

3.4.2: Application to our problem and results: We decided to use the R-package randomForest [8] since MATLAB statistics toolbox did not seem to support Random Forests. The important parameters to optimize for both types of Random Forests were *m* and *MinNodeSize* (for each of the trees grown). In order to optimize these parameters we drew bootstrap samples of size N_b=20000, grew 1000 trees and estimated effectiveness of the parameters on held-out cross validation data with AUC as the metric. We found optimal values for *m* and *MinNodeSize* by using brute-force search over a limited search space. We used the Random Forest with optimized parameters to predict class probabilities for the test data.

We found that attempts at "cleaning" data like saturating the Ruul parameter or replacing coded values with medians backfired. Since the package randomForest in R, did not handle missing values gracefully, we started by training three different models similar to SVM's. We did this for both regular data and balanced data (created using under-sampling) and used both Regression and Classification based random forests but were unable to significantly improve on a simple reference solution that used Random Forest regression on only the clean input features. We achieved a key breakthrough by replacing missing values with coded values (-1 was used) and training, optimizing a single model using all 10 input features. We also gained some improvement by using over-sampling of class-1 training records to mitigate class-imbalance as opposed to under-sampling of class-0 records. Using Random Forest regression, replacing missing values with -1's and over-sampling class1 records we were able to get an AUC_PART of 0.86203 which was our highest.

4: Discussion: Logistic Regression, Probit Regression and SVM's are traditionally not very effective when there is a significant amount of missing data, as in the case of our problem [6]. The non-competitive performance of Logistic and Probit regression can also be attributed to the fact that, they produce linear decision boundaries using only 10 input features even though there is an abundance of training data (150,000 records). As for SVM's, they are not specifically encouraged while training to optimally predict class probabilities. Rather the goal is to maximize separation from the separating hyper-plane in the higher dimensional space. Significant missing data along with this fact might explain the relatively lower performance of SVM's on this dataset.

Classification trees are effective at handling missing data and outliers. They are also invariant to monotone transformations of an input feature. This might explain why Ruul parameter saturation, which produces significant benefits for Logistic, Probit Regression and SVM's, is not necessary for Classification trees. Classification trees are notorious for having high variance [6] and we were pleasantly surprised that a well-tuned Classification tree could out-perform tuned C-SVM's.

Random Forests were invented as a method to reduce the Variance problem of Classification trees and it is no-surprise that they out-perform Classification Trees. We were able to obtain significant benefits by tuning *m* (the number of input features selected at random at each node as split candidates) and the minimum node size. However we are unable to satisfactorily explain the following results:

- Random Forest regression was better at predicting class probability estimates than Random Forest Classification. Our best guess is that this might have to do with the split criterion - Mean square error, used for regression as opposed to the Gini Index that is used for Classification.
- The hack of replacing missing values with -1 performed significantly better than a multiple model approach, where depending on the clean features available in the test-record, an appropriate tuned Random Forest was used.

We tried three approaches to handle the imbalance between class0 and class1 records - under-sampling of class-0 records, over-sampling of class-1 records and using different weights for class0 and class1 records when the algorithm permitted it. We obtained best performance with over-sampling of class1 records.

Table2: Final summary

Approach	AUC_PART	AUC_FULL
Logistic regression, multiple models, Data cleaning and under-sampling	0.84919	0.855270
Probit regression, multiple models, Data cleaning and under-sampling	0.84920	0.855333
SVM with RBF Kernel, multiple models, Data cleaning and under-sampling	0.85013	0.856121
Classification trees using Optimal *MinLeaf*, Twoing for splitting and surrogate variables	0.85310	0.857019
Reference Solution using Random Forest Regression with only "clean" features	0.85976	0.864388
Random Forest classification using coded values for missing values and under-sampling	0.85842	0.864954
Random Forest regression using coded values for missing values and over-sampling	**0.86203**	**0.867125**
The Winning Team (Not us, we were 135th out of 970 teams)	***NA***	***0.869558***

5: Conclusion: Based on our experience in this project, given a binary classification problem with missing data, we would recommend first trying Random Forests, tuning *m* and minimum node size and addressing class imbalance by over-sampling. After the results were announced we learned that the best scores were obtained by blending/averaging the results from different schemes like Random Forest Regression and Classification, Gradient Boosted Regression and Classification Trees and Neural networks. Also the winning team combined the 10 given features to produce as many as 35 features to use in their algorithms.

References:

1: http://www.kaggle.com/c/GiveMeSomeCredit
2: Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1--27:27, 2011.
3: H.-T. Lin and C.-J. Lin: A study on sigmoid kernels for SVM and the training of non-PSD kernels by SMO-type methods. Technical report, Department of Computer Science, National Taiwan University, 2003.
4: John C.Platt: Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods (1999). Advances in Large Margin Classifiers.
5: http://www.mathworks.com/help/toolbox/stats
6: Hastie, Tibshirani and Friedman: The Elements of Statistical Learning: Data Mining, Inference and Prediction, 2nd Ed. 2009
7: http://www.stat.berkeley.edu/~breiman/RandomForests
8: A. Liaw and M. Wiener (2002). Classification and Regression by randomForest. R News 2(3), 18--22.