CS578 Preliminary report: Loan Default Prediction

Priyank Jain: jain206@purdue.edu

November 2, 2016

Abstract

Can we predict whether or not it is safe to make a loan to a person based on his or her transaction history? In 2014, researchers at Imperial College London sponsored a competition challenging participants to predict the quantity of loss for a certain set of loans. In this project, I propose to simplify things by looking only at whether or not a loan will default.

1 Introduction

My goal in this project is to use the features of a loan to predict whether that loan would default or not. Such an application can be tremendously useful in banking, since banks make money by lending money to debtors. Identifying loans which might default can tremendously help banks to avoid loss and avoid unnecessary trouble/legal procedures.

2 Dataset and Technology

The training and test data files consist of 105471 and 210944 independent samples respectively, each with 779 anonymous features including the label. It has already been de-trended, randomly sorted, and standardized. The data set was suggested on the class website and can be found at: https://www.kaggle.com/c/loan-default-prediction

I use Python programming language and Git for Version Control. The code used for experiments could be found here: https://github.com/priyankjain/loan-default-prediction

Scikit python library is used for training and other machine learning related pipeline. Scikit is a popular machine learning library and has implementation of most machine learning algorithms. The code is written in object-oriented style. Charts are generated using matplotlib library in Python.

3 Experiment 1: Using binary SVM

3.1 Setup

In this experiment, I use binary SVM for prediction. Since the dataset is huge, I take a subset of the dataset for the purpose of this experiment. I take first five 2500 positive samples from the training file (for which there is no default) and 2500 negative samples from the training file (for which there is default). The various parts of the pipeline are explained below.

3.2 Data Cleaning

The last column of each row in the data file corresponds to the loss the bank faced for that particular loan. If the loss is 0, it implies the loan did not default. For our binary classification task, I convert this loss column from continuous feature to 0-1 feature. If the loss column is non-zero, I simply assign it label 1, implying that it corresponds to a default. The first 2500 positive and negative samples are then picked up from the training data file for training the binary SVM. Any row containing a 'NA' value is dropped, since it corresponds to a missing value. Before continuing with rest of the pipeline, these 2500 positive and 2500 negative samples are shuffled.

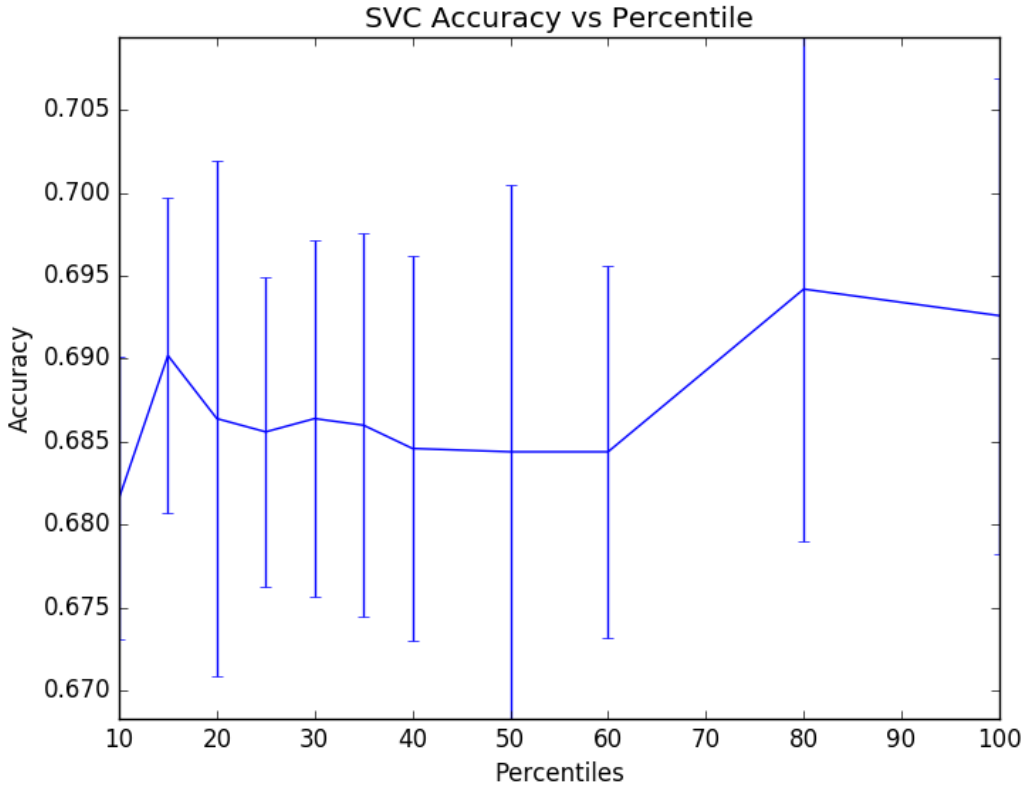

Figure 1: Variation of accuracy with percentile of features selected to train the binary Support Vector Classifier

3.3 Feature Selection

I use univariate feature selection for this experiment. Since training a SVM is a quadratic programming problem, using greedy subset selection can be a computationally expensive problem. Hence, I use univariate feature selection using mutual information. I select top k percentile of the features and then train the SVM using those features. Figure 1 below depicts how the accuracy of binary SVC varies according to the percentile of features selected. The result says that there is not much incentive in using more features. Hence I select features based on the percentile k which gives me a decent enough accuracy (not necessarily highest), and which makes the algorithm run fast (since it's computationally cheaper).

3.4 Normalization

Since all features have different scales, I use standard normalization technique. This technique converts the distribution of each feature to have unit variance and zero mean.

3.5 Training

I use sklearn.svm.SVC class for training the classifier. I use radial basis kernel for now. This class needs to be initialized with a particular C, where C is the penalty for violating the margin. I select C on the basis of cross-validation. Figure 2 shows the variation of accuracy of the support vector classifier as the slack penalty in increase as from 0.01 to 30. Value of changes as follows: 0.01, 0.03, 0.1, 0.3, ..., 30. The figure says that C = 3 gives the best accuracy. The results are a bit different for different runs, but generally C=1 or C=3 gives the best result for the purpose of this experiment.

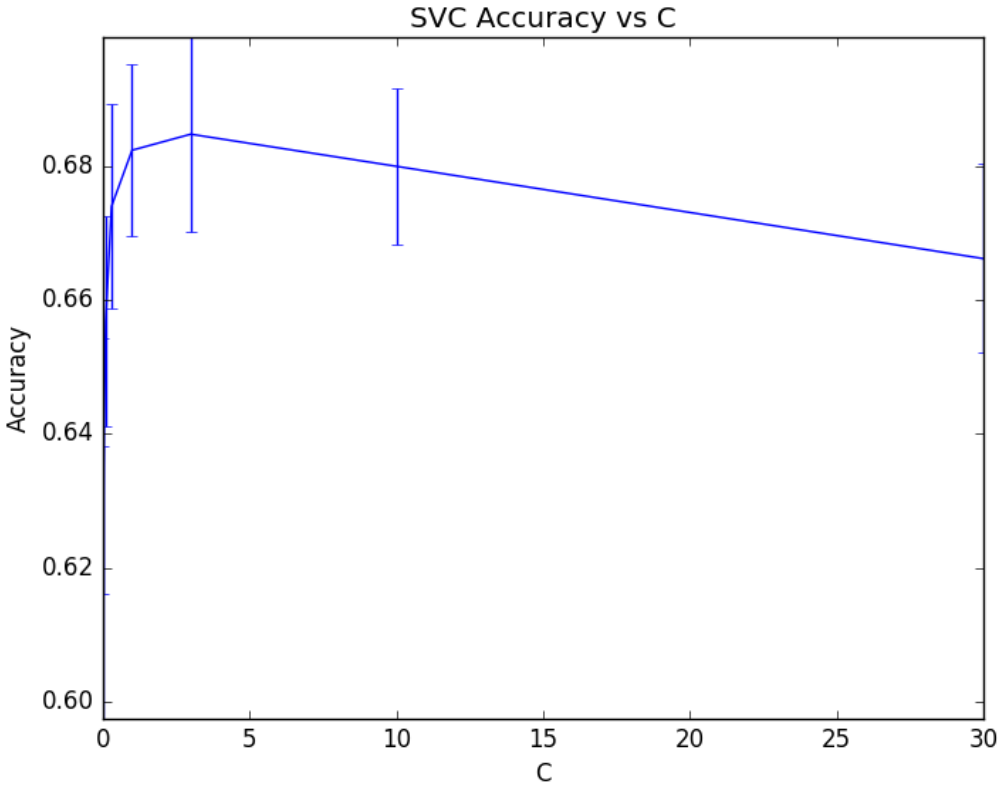

Figure 2: Variation of accuracy with slack penalty C selected to train the binary Support Vector Classifier

3.6 Cross-validation and accuracy

I use k-fold cross validation. Here k=5. Code for cross-validation could be found in code/CrossValidationModule.py. Accuracy is reported as average of the accuracies over the five different classifiers. The accuracy is reported as the number of correct predictions divided by the total predictions. The accuracy is reported for the test set in the cross-validation step rather over a seperate set. This would be changed in the final project report.

4 Future work for final project report

If I get access to a computing cluster offered by Purdue, I'll report results against the entire dataset rather than a subset. I would involve better measures of accuracies, like precision and recall. I would run another experiment to see the performance of a one-class support vector classifier. Final report would include ROC curves and variation of error as the size of the training set increases. I would potentially include SVDD and run experiments with variations of kernel.