Capstone Proposal for Machine Learning Nanodegree

By Shobhit Mishra

Definition

Project Overview

Financial companies who provide loans to their customers face risk of defaulted loans. Companies often collect a vast amount of data about their customers. Using Machine Learning techniques, we can predict the outcome of a loan. We can analyze the data to identify the relevant features and use this data to train the Machine Learning model. Having a reliable and accurate model reduces the risk of loan default significantly.

In this project, I use the publically available loan data of Lending tree to predict the loan outcome. I clean up the data and do exploratory data analysis to identify the useful features. I then create a model to predict the loan outcome. I use the f1 score as a metric to evaluate the performance of the model.

Problem Statement

The data provided by Lending tree is rich and has multiple loan status but can reduce the loan prediction problem to a binary classification problem. The goal is to build a loan classifier which takes various features of a loan application as input and predicts if the loan will default or not. The project involves the following steps:

1) Download the Lending tree loan data
2) Get rid of empty columns and rows
3) Do exploratory data analysis to identify the trends and useful features
4) Do data cleaning and transform categorical data to numerical data.
5) Split the data in train and test set
6) Train a model on the training data set
7) Predict the outcome on test data set and measure the model's performance.

Metrics

We can use several metrics to measure the performance of binary classification models. Accuracy is one of the most intuitively understood metrics. Accuracy is the ratio of correctly identified labels and total number of labels. Although accuracy seems to be a natural measure of performance, it is not suitable for a skewed data set. e.g. in our dataset, if we categorize every loan as paid, the accuracy will come out to be 0.86 which on surface looks very good. The classifier however will still be very dumb and will classify every application as loan worthy. That is not what we want.

Other two possible metrics are precision and recall. Precision is the ratio of "correctly identified positive" and "overall predicted positive". The recall on the other hand is the ratio of "correctly identified positive" and "actual positive". If the classifier is biased towards getting positive, it will have good recall score but poor precision score. On the other hand, if it is biased towards predicting negative outcome then it will have good precision but poor recall score.

Due to aforementioned limitations, neither precision nor recall are very good metrics. A good metric should take both precision and recall into consideration.

One such metric is F1 score. F1 score is a widely used metric to measure the performance of binary classifiers and that is what I am going to use in this project. F1 score considers both precision and recall of the model. According to Scikit documentation (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html#sklearn.metrics.f1_score), F1 score can be defined as:

```
F1 = 2 * (precision * recall) / (precision + recall)
```

We'll use the scikit's built in method to calculate the f1 score.
Another possible metric is G measure. While F1 score is the harmonic mean of precision and recall, G measure is geometric mean of precision and recall.

Analysis

Data Exploration

The lending tree dataset is an unprocessed data set with hundreds of columns. The dataset is so big that lending tree provides a separate excel sheet to explain what individual columns stand for. I have attached that excel sheet (LCDataDictionary.xlsx) in the root folder. Here is the summary for the unprocessed data set:

```
RangeIndex: 42542 entries, 0 to 42541

Columns: 111 entries, id to total_il_high_credit_limit

dtypes: float64(86), object(25)

memory usage: 36.0+ MB
```

Out of these columns, 60 columns have either blank or only one value. These columns are of no use to us and we can safely drop them.

The 'loan_status' column stores the outcome of loan. This column has several possible values. Here are the possible loan_status values:

```
array(['Fully Paid', 'Charged Off', 'Late (31-120 days)', 'Current',

       'Late (16-30 days)', 'In Grace Period', 'Default', nan,

       'Does not meet the credit policy. Status:Fully Paid',

       'Does not meet the credit policy. Status:Charged Off'], dtype=objec
t)
```

The respective counts are as follows:

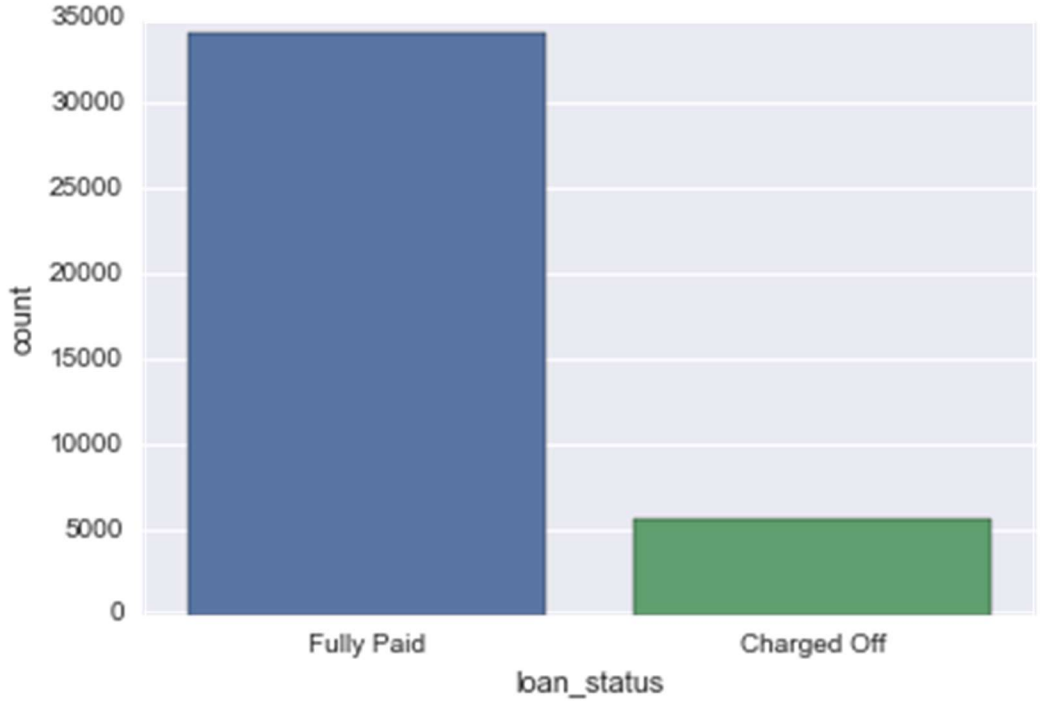
```
Fully Paid                                                34108

Charged Off                                                5662

Does not meet the credit policy. Status:Fully Paid         1988

Does not meet the credit policy. Status:Charged Off         761

Late (31-120 days)                                           10

Current                                                       3

In Grace Period                                               1

Late (16-30 days)                                             1

Default                                                       1
```

We'll consider only "Fully Paid" and "Charged Off" for our project. We'll drop the rows with other values to simplify our analysis.

There are too many columns in the dataset and it is not possible to describe what individual columns stand for. Please take a look at the 'LCDataDictionary.xlsx' in the root folder if you want to understand what individual columns stand for.

Exploratory Data Analysis

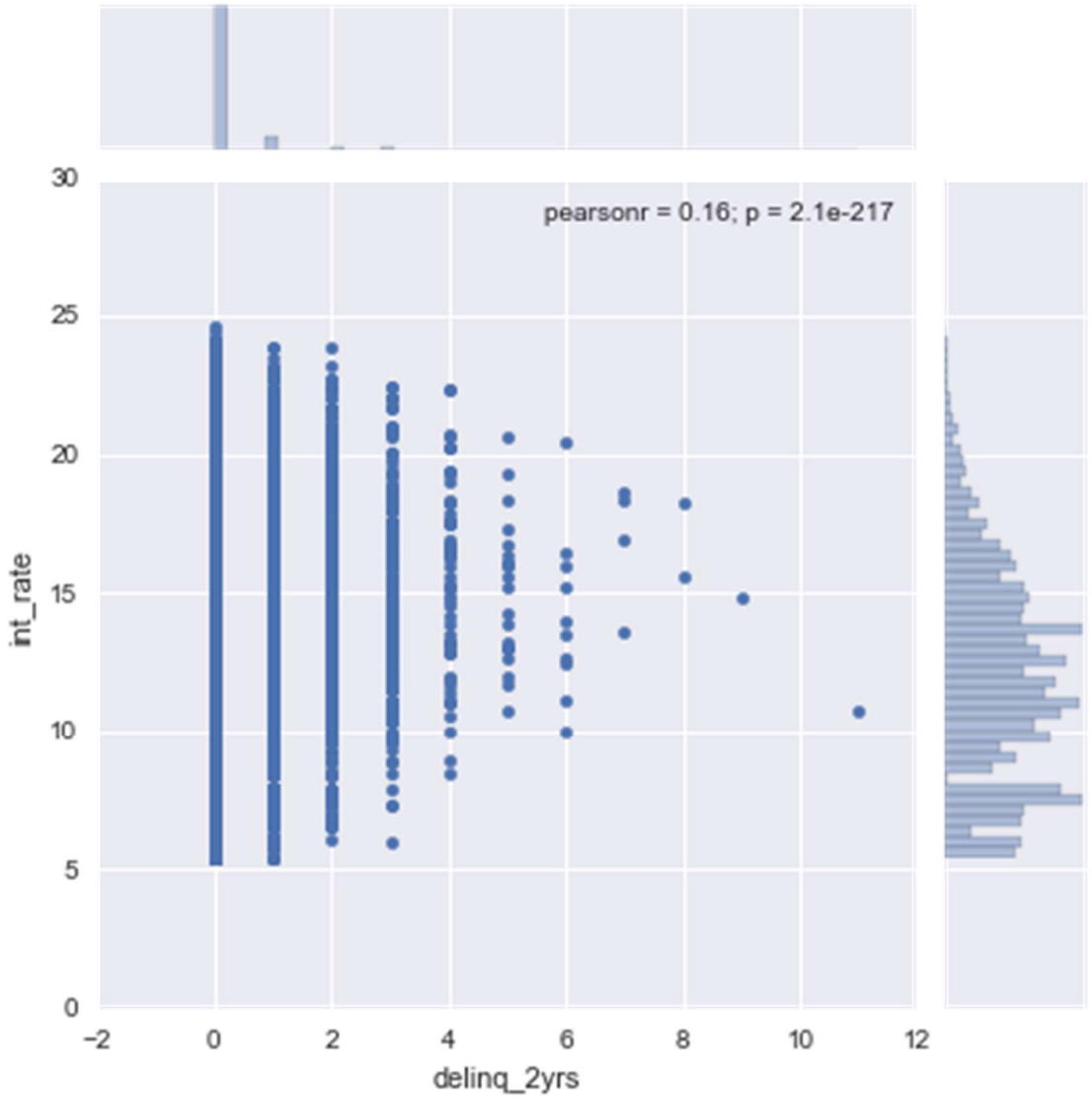
First, we want to see the ratio of 'paid loans' and 'defaulted loans' to see if the data is skewed. Here is a count plot of the loan_status.



As is evident from the count plot, the data is skewed and only 14% loans are charged off. Therefore, any classification model should at least have 86% accuracy.

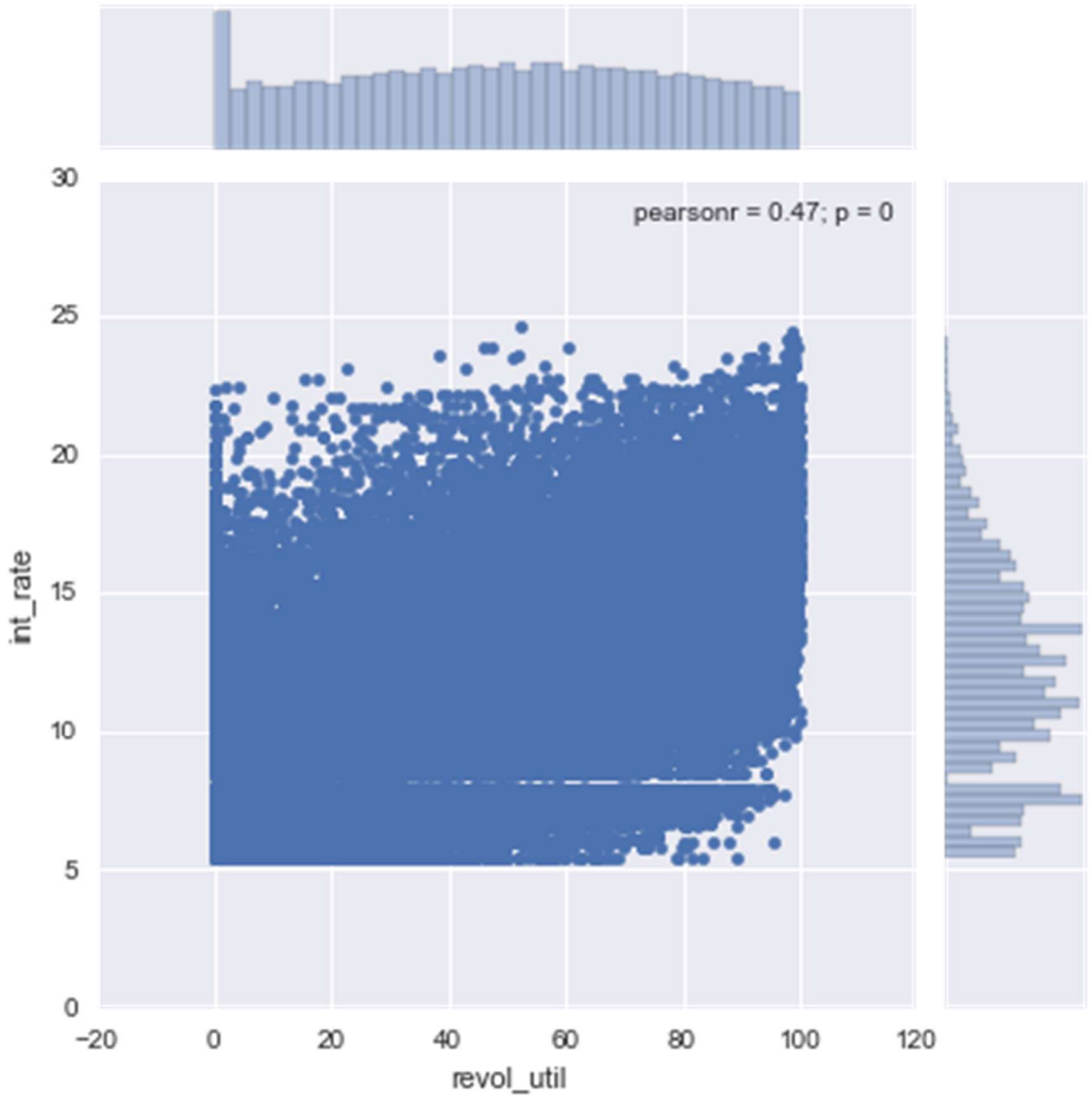
A higher risk loan has a higher interest rate. I tried to explore the relationship of interest rate with other attributes. The following plot is a join plot between interest rate and delinq_2yrs.

Note: Delinq_2yrs denotes the number of 30+ days past-due incidences of delinquency in the borrower's credit file for the past 2 years.



As we can see, the interest rate seem to start at a higher rate for higher delinquency rate. There are however other factors which affect the interest rate and that is why there is no clear linear relationship between these two features.

The following graph is a joint plot between interest rate and 'revol_util'. 'Revol_util' denotes "Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit."



Here also there doesn't seem to be any direct relationship between int_rate and revol_util. The revol_util seems to be uniformly distributed (look at the top portion of the graph).

I did quite a bit of exploratory data analysis by plotting several graphs but didn't find anything significant. It seems that the outcome doesn't depend directly on a few features but is a combination of multiple factors.

Algorithm and Techniques

Since this is a binary classification problem, we can choose from a variety of algorithms. I used this (http://scikit-learn.org/stable/tutorial/machine_learning_map/) map from scikit documentation to narrow down my choice of estimators. Our sample size is close to 40K. According to this guide, we

should start with Linear SVC and if that does not perform well, move to SVC or Ensemble classifiers. I adopted the same strategy. I first tried Linear SVC and then tried RandomForest Classifier. I found Linear SVC adequate enough for our data set.

Linear SVC is a support vector machine classifier with a linear kernel. Support vector machines try to represent the data points in space and then divide them in separate categories. SVM tries to keep the gap between two categories as wide as possible. Sometimes, it is not possible to linearly divide the data in separate categories. In such cases, the SVM maps the data in higher dimensions using kernel trick and then divides it into separate categories. The kernel trick uses a mathematical function to transform the original data to a hyper dimension space. Some examples of kernels are RBF kernel, Polynomial kernel, Fisher kernel (https://en.wikipedia.org/wiki/Kernel_method#Mathematics:_the_kernel_trick). Linear SVC is a SVM classifier with a linear kernel. Linear SVC tries to classify the data linearly without any transformation in the hyper dimension. Here are the tunable parameters in Sci-kit implementation of Linear SVC (according to Scikit documentation):

- Penalty. Specifies the norm used in penalization. Default is 'l2'
- Loss: Specifies the loss function. Possible values are string, 'hinge' or 'squared_hinge'. Default is 'squared_hinge'
- Tolerance: tol for stopping criteria. Default is 1e-4.
- Penalty parameter C. Default is 1.0
- Multi class: This is used for multi class classification. It doesn't apply in our case
- Fit intercept: Boolean to indicate if the model should calculate the intercept. Default is true.
- Intercept scaling: Adds a synthetic feature to the instance vector. The default value of synthetic feature is 1.0.
- Class weight: Used to tweak the weight of a certain class. Default value is balanced which assigns weight 1 to every class.
- Random state: Seed for random number generator
- Max iteration: Maximum number of iterations to run. Default is 1000.

In our model, we use default values for each parameter. We tune the parameters if the performance of the model is unsatisfactory.

Random forest classifiers are an ensemble learning method. Ensemble methods use multiple learning algorithms to get a better performance than what any one algorithm can achieve on its own. Random forests create a number of individual decision trees. Random forests chooses mode of the individual trees as the final category. For example, if there are 10 decision trees in the random forest and six of them predicts category A and 4 predicts category B then random forest will chose category A as the final answer. In scikit, we can choose the number of trees in the forest by setting the value of n_estimators. Random forest classier in sci-kit has a number of tunable parameters but default values work very well in our case.

Benchmark

As we saw in the exploratory data analysis that our data mostly belongs to "Loan paid" class. Only 14% of the data belongs to "Charged off" category. If we know the count of each category, a naïve classifier

will put the data to “Loan paid” category 86 times out of hundred. Our classifier should perform significantly better than the naïve classifier without overfitting. We have also discussed the problems in using “accuracy” as a metric.

Due to the limitations of precision and recall, we are going to use f1 score as a metric for out model’s performance. I tried to find similar model’s to set a benchmark for f1 score. After much struggle, I found two such implementations.
In the first implementation (http://www.wujiayu.me/assets/projects/loan-default-prediction-Jiayu-Wu.pdf) the author uses the lending tree data itself. The analysis was done in 2014. He got the following scores for his model.
Accuracy=0.7395, Precision=0.2789, Recall=0.6498 F-measure=0.3903
As we can see, the model gets a decent accuracy score but bad precision and very poor F1 score.

Another such implementation is at
http://cs229.stanford.edu/proj2014/Kevin%20Tsai,Sivagami%20Ramiah,Sudhanshu%20Singh,Peer%20Lending%20Risk%20Predictor.pdf
They however try to maximize the precision. Their top f1 score is around 0.8 on LibSVM model and the top precision is 0.95.

I also found a similar dataset on Kaggle (https://www.kaggle.com/c/loan-default-prediction) but that was not a classification problem. It was a regression problem and the metric was MAE (mean absolute error).

Looking at these examples, I think a f1 score of 0.8 with a precision of 0.9 should be desirable.

Methodology

Data Preprocessing

The loan data is a real world data and the real world data tends to be unclean and messy. I did quite a bit of data clean up before training the model. Below are the data cleanup steps:

- Dropped all the columns with less than two values.
- Studied the data and dropped the columns which didn’t seem to directly relate to the loan outcome. One such column is loan description. This step was mostly experimental. I erred on the side of having more data. I didn’t drop the column if it seems even remotely related to the loan outcome.
- There were six possible loan outcomes. I reduced it to two. I did it to reduce the problem to a binary classification problem. The other outcomes had very few value counts.
- I dropped the rows with empty values. I could do it safely because there were very few rows with empty values.

- The term, interest rate and revol_util are numerical types but the actual data was string. Interest rate for example was given as 10%. The '%' character is of no interest to us. I cleaned up these columns to have only the numerical part and converted the string to float.
- Loan_status and Verification_status were categorical values. I converted them to numerical values using one hot encoding technique.

After all the data cleanup steps, the data summary looks as follows:

```
Data columns (total 17 columns):
loan_amnt               39023 non-null float64
funded_amnt             39023 non-null float64
funded_amnt_inv         39023 non-null float64
term                    39023 non-null float64
int_rate                39023 non-null float64
installment             39023 non-null float64
annual_inc              39023 non-null float64
dti                     39023 non-null float64
delinq_2yrs             39023 non-null float64
inq_last_6mths          39023 non-null float64
open_acc                39023 non-null float64
revol_util              39023 non-null float64
total_pymnt_inv         39023 non-null float64
pub_rec_bankruptcies    39023 non-null float64
Source Verified         39023 non-null float64
Verified                39023 non-null float64
Fully Paid              39023 non-null float64
```

We have 17 columns and 39023 rows.

Implementation

The majority of work in this project was around data cleanup. The model training and evaluation was relatively straight forward. I used the Scikit estimator map at this (http://scikit-learn.org/stable/tutorial/machine_learning_map/) link to narrow down the list of classifiers. I decided to try several classifiers with default values and compare their results. All my implementation is in Lending_Tree_Loan_Prediction.ipynb in the root folder.

I split the data in train and test set before training the model. The split size is 0.3. Below is the list of all the classifiers and their corresponding results for the test set. I didn't get the scores for the training set because that is not very meaningful.

1) Logistic Regression: I trained the logistic regression model with default parameters. The logistic regression performed surprisingly well. Here is the classification report for this model on test set:

	precision	recall	f1-score	support
0.0	0.98	0.76	0.86	1570
1.0	0.96	1.00	0.98	10137
avg / total	0.97	0.97	0.96	11707

2) LinearSVC: I trained the Linear SVC with all the default parameters on the same training data set. Here is the classification report for this classifier on test set.

	precision	recall	f1-score	support
0.0	0.87	0.84	0.85	1570
1.0	0.98	0.98	0.98	10137
avg / total	0.96	0.96	0.96	11707

3) Random Forest: I trained RandomForest classifier on the same training data set with 200 estimators. I also tried with 600 estimators and the results were the same.

	precision	recall	f1-score	support
0.0	0.99	0.77	0.87	1570
1.0	0.97	1.00	0.98	10137
avg / total	0.97	0.97	0.97	11707

It is quite evident from the results that the best performing model is the Random Forest classifier but other classifiers are quite close.

Refinement

I used the default parameters for all my classifiers. I tried different number of estimators for the Random Forest classifier but the results didn't improve. I got very consistent scores for all my models with different parameters.

One possible refinement is choosing different columns in the data set. I experimented with several columns and picked the most suitable ones.

Results

Model Evaluation and Validation

As we saw in the previous section, both LinearSVC and Random Forest performs extremely well on the classification without overfitting. The default parameters worked very well for me and the performance didn't improve with any parameter tuning. I wanted to see if the model's performance is sensitive to the data. I tried various split sizes (0.2 to 0.4) and the performance didn't change significantly.

Lending tree has been adding to this data set for several years. It is quite possible that their loan prediction model has been improving over time. The unpaid loans passed through the process despite the scrutiny.

Justification

Number of charged off loans is almost 0.14 (or 14%). The f1 score of the random forest classifier is 0.97 on a test set of 11k. It is quite clear that the model can outperform an educated guess by a huge margin.

The f1 score for the model is also significantly higher than the benchmark score. This model performs better in both, f1 score and the precision, than the other two models I referenced in the benchmark section. Those studies were conducted a few years ago and the data size has grown significantly since then. More data can explain (at least partially) the difference between the performance of these models. It is noteworthy that lending tree has a separate dataset about rejected loan applications. It is not clear what criteria is used to reject an application but the current dataset is a filtered dataset without a lot of noise.

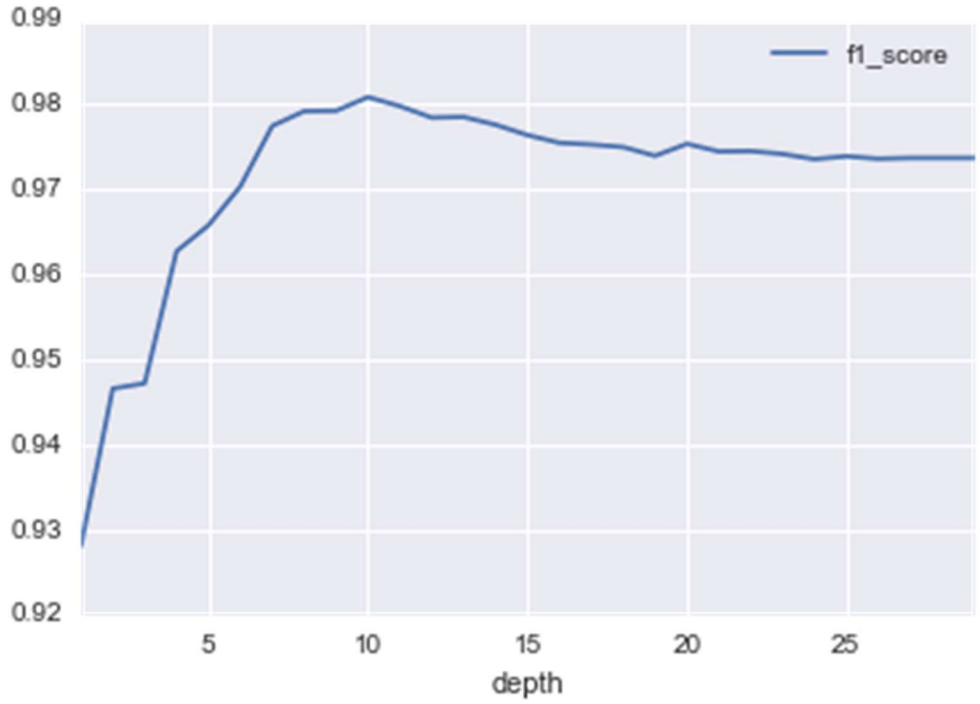
In short, the model can be used to predict the loan outcome on a clean dataset.

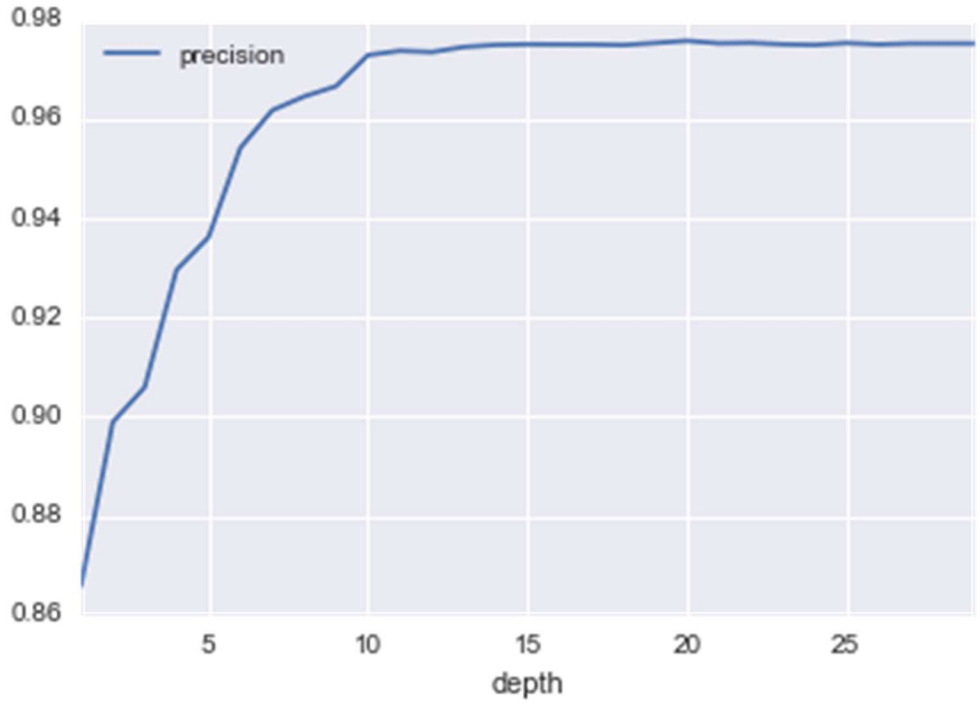
Conclusion

Free-Form Visualization

This was an interesting part of the model exploration. From the beginning, I wanted to see if there are certain features which plays a prominent role in deciding the loan outcome. I learned that a decision tree visualization can give me a great insight in the relationship of certain features and the loan outcome.

Before doing that, I wanted to see how the f1 score and the precision score changes with tree depth. The following plots show the trend of f1 score and precision with tree depth.

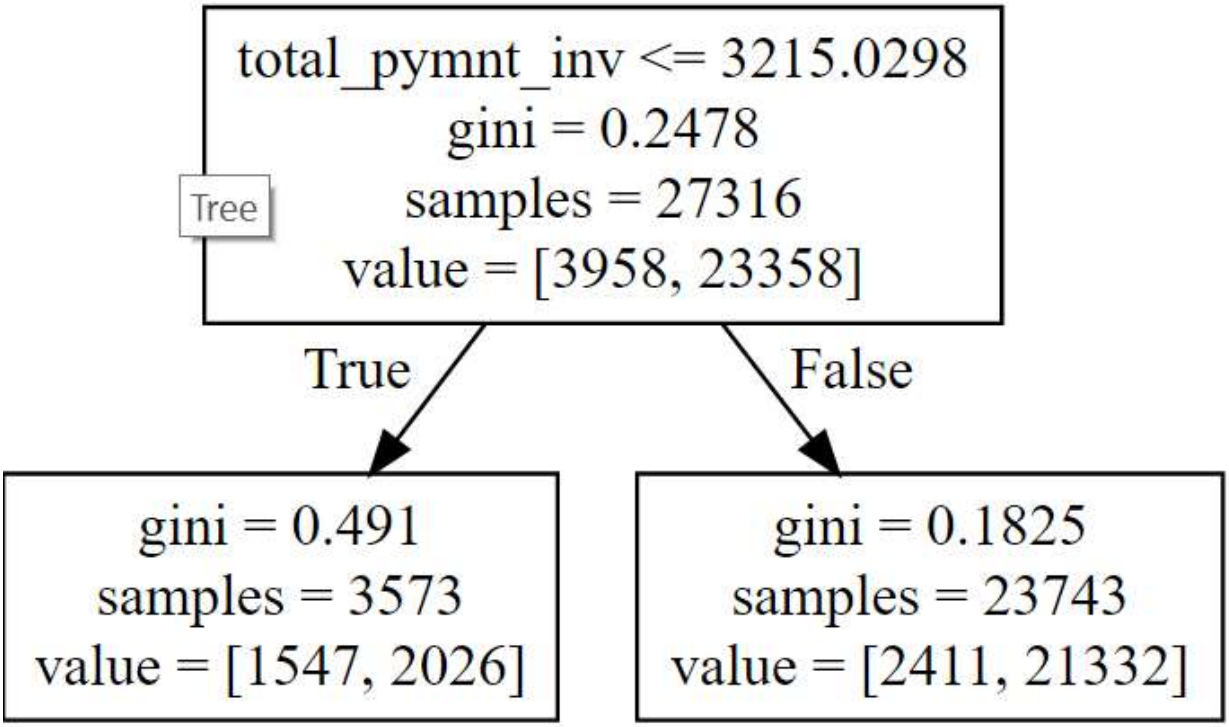




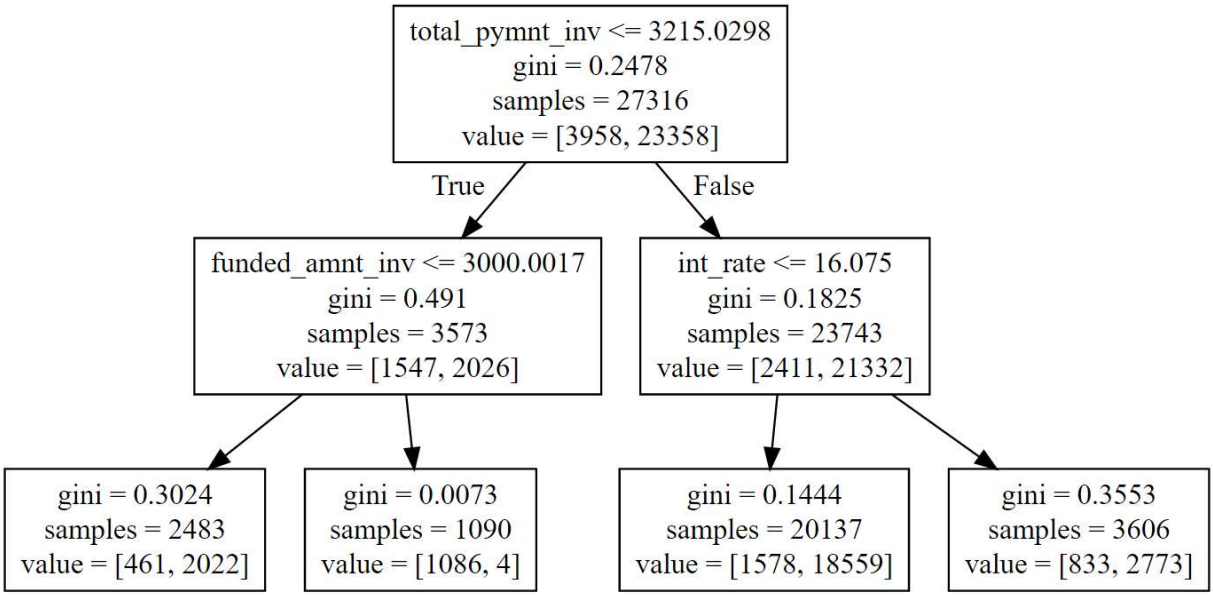
As we can see the, the f1 score improves with depth and then starts decreasing. It is noteworthy that the f1 score is very high in the beginning itself. It is clear that there are some features which play a very prominent role in predicting the loan outcome.

To get the decision tree visualization, I generate the dotfile using graphviz library and then use http://webgraphviz.com/ to generate the visualization from the dotfile.

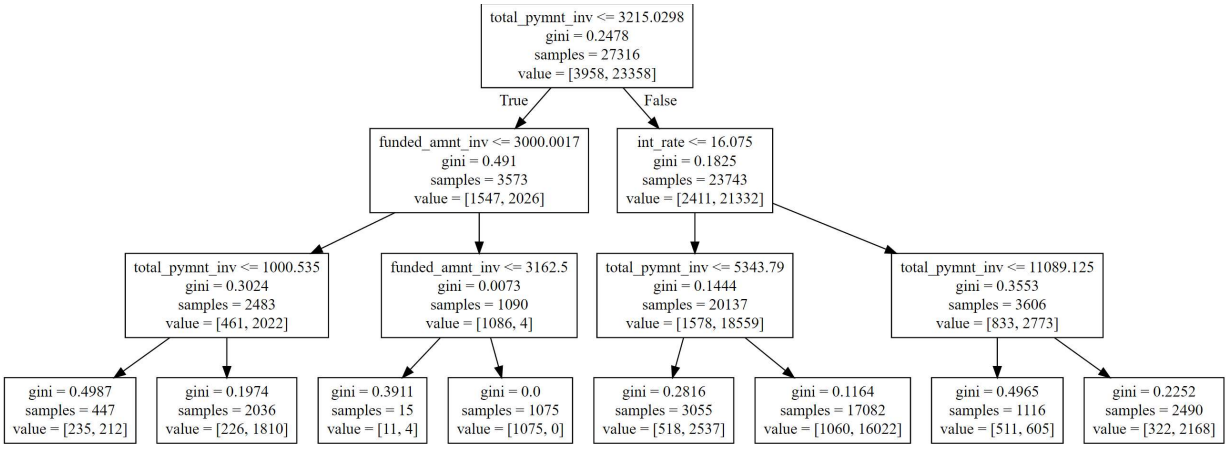
Below is the visualization for depth 1:



For depth 2, the visualization is as follows:



For depth 3:

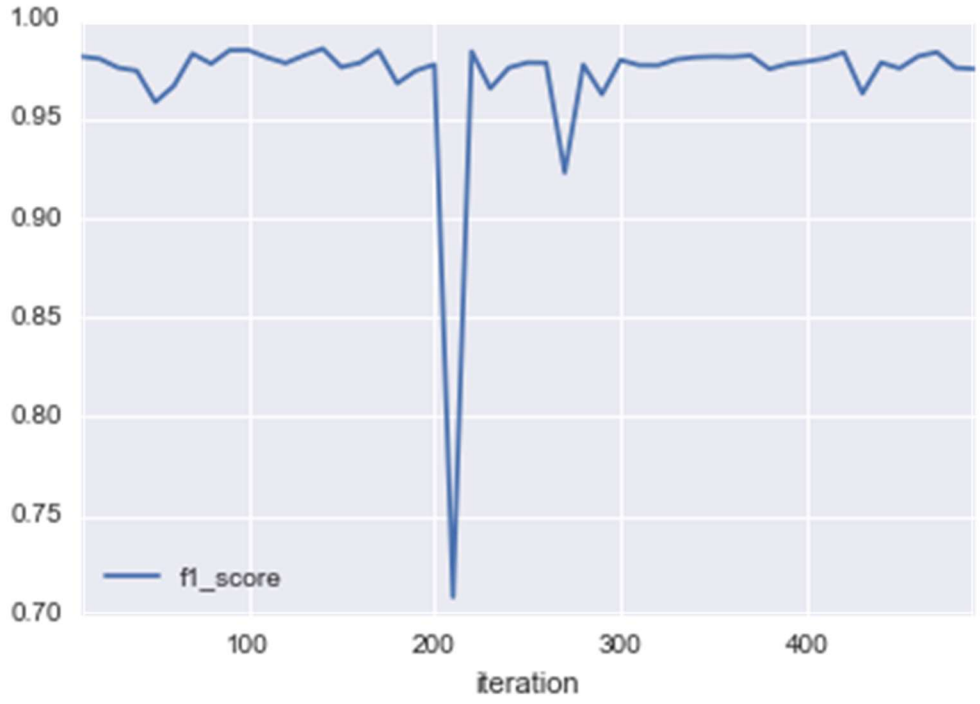


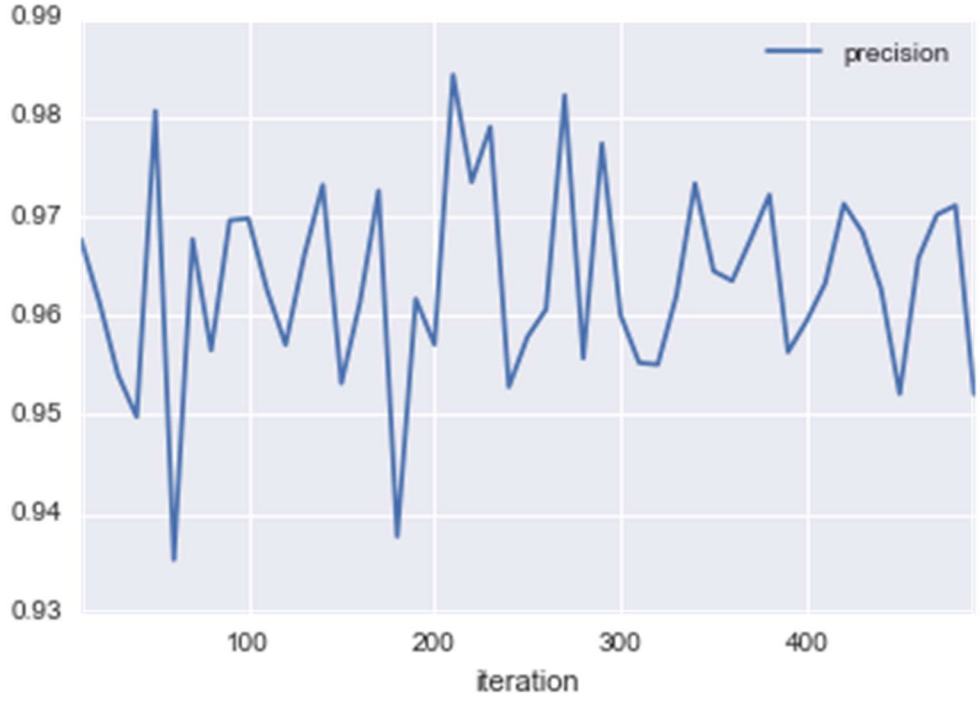
It is not possible to clearly show the visualization for depth more than 3. I have attached the dotfiles for up to depth 30 in the folder “Plots and images\Dotplot files”

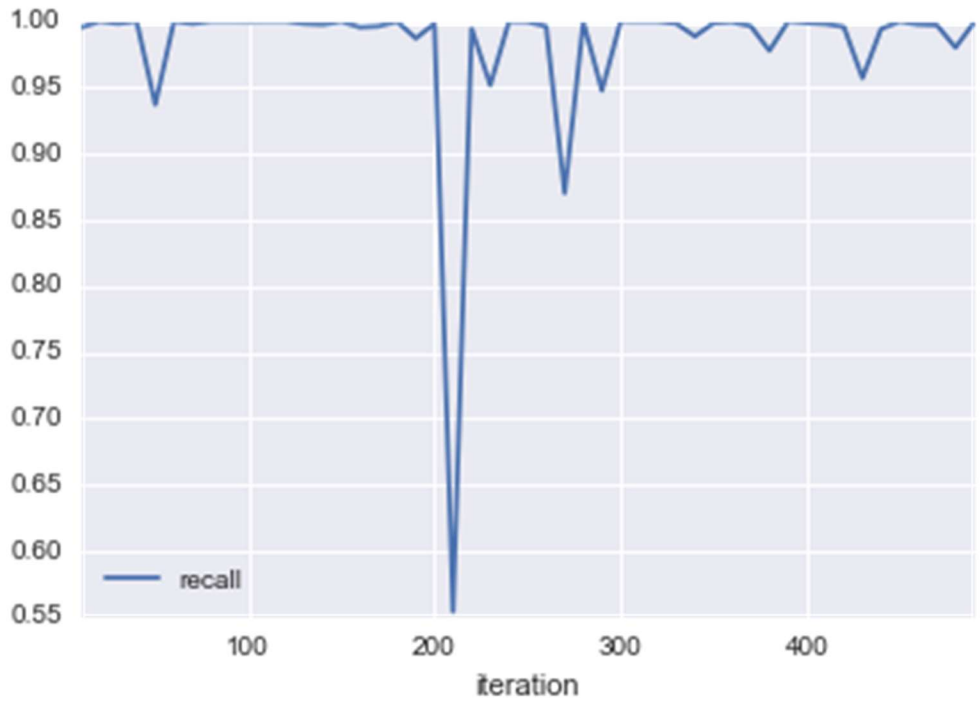
By looking at the visualization, it appears that the following features play a major role in predicting the outcome.

total_pymnt_inv: Payments received to date for portion of total amount funded by investors
int_rate: Interest Rate on the loan
funded_amnt_inv: The total amount committed by investors for that loan at that point in time.
I am not sure why they play an important role in predicting the outcome. Perhaps a person with more domain knowledge can make sense of it.

I did a similar analysis on the linear svc model. I changed the value of max iteration in a loop and plotted the value of f1 score, precision and recall with iteration. For most part the results are not remarkable but there is one huge dive for the recall score when the number of max iteration is around 210. I am totally befuddled why we see that. Here are the plots:


1.00
0.95
0.90
0.85
0.80
0.75
0.70
f1_score
100
200
300
400
iteration


0.99
0.98
0.97
0.96
0.95
0.94
0.93
precision
100
200
300
400
iteration



Notice the dip in the values for both the f1 score and the recall score. F1 score dip can be explained by the dip in the recall score. I am not sure why recall drop to 0.55 from 1.0 at iteration 210.

Reflection

The project involved the following steps:

1) Identify the problem domain and find a relevant dataset
2) Download the data and do a cursory analysis
3) Clean up the data
4) Do exploratory data analysis to find out the relevant features
5) Extract the relevant data.
6) Find a suitable classifier
7) Train the classifier on training data
8) Run the classifier on test data and measure the performance

The step 4 and 5 were the hardest for me and I did several iterations of both steps. I am not familiar with financial jargons and have little to no idea what features make an application loan worthy. I tried to find information on Internet but to no avail. Finally I resorted to educated guess and refinement to achieve my goal.

The data clean up and extraction was also the most interesting part of the project. This was the first time when I picked a real world problem and went through the entire process of cleaning, processing and prediction. I also looked at some of the kaggle datasets as an alternative. I found that the kaggle datasets are much cleaner and organized as compared to the Lending tree dataset. This project gave me

the confidence to explore the publically available data on my own and to dive deep. I also learned that the experimentation and iterative refinement plays a big role in Machine Learning.

Improvement

The lending tree data set was skewed. The successfully paid loan samples dominated. The number of charged off loans were only 14%. It would be interesting to see how the model performs if we remove enough “paid off” loan samples to make the ratio 1:3 i.e 75% of one category and 25% of the other category.

It would also be interesting to apply the same methodology on other data sets and find the common patterns. Perhaps there are some features which appear in each data set and pay crucial role in predicting the loan outcome.

I did little to no parameter tuning for my models. Perhaps we can get better results with more parameter tuning.