Hong Seh Looi
frank@minerva.kgi.edu

Report on the Lending Club Machine Learning Model

Variables included in the model

The data that is fed to the model is combined from 2 spreadsheet sources, loan applications that are accepted and those that are rejected between 2007 and 2011. Each source had different numbers of column including different components, but only some were chosen for the model. Specifically, the Rejected Data only had 9 components per applicant while Accepted Data had 145.

The variables included from the Rejected Data are Loan Amount, Loan Title, Debt-To-Income Ratio (DTI), State, Employment Length and Policy Code. Application date was excluded because both data sources are from 2007 to 2011, and a Chi-squared test showed only 0.2 correlation between application date and the acceptance of a loan. This means that Application Date does not have a strong predictive power on the outcome of a loan. Zip code is also excluded from the data because any variance in zip code is already explained by variance in State, therefore the predictive variables are overlapping and one of them can be omitted. Finally, Risk Score is not included because Accepted Data does not have this feature, which makes concatenating difficult. For Accepted Data, a similar set of predictive variables were omitted for the same reasons. However, most Accepted Data were only relevant to terms of the approved loan, such as interest rate, payment term, and payment made, rather than the outcome of the loan acceptance versus rejection.

Cleaning and transformation on the data

The variables that were excluded as mentioned above were dropped from the data that is fed to the model. This reduced the number of dimensions in our data while preserving predictor variables that have the highest predictive power (ie minimal loss of information). Furthermore, data was also renamed into short forms (eg: ‘Debt-To-Income Ratio’ to ‘dti’) to ease programming the model.

Categorical variables are also transformed into dummy variables. Categorical variables such as different loan titles are enumerated into different categories, expressed by numbers, to make sure the model ran successfully. Additionally, we added a column for “Approval” after combining the two datasets which identified loan applications that were approved from those that were rejected. These became labels for training and testing for our machine learning model. Finally, data types were converted between integers, strings and floats to ensure conformity with the functions that were called within the model.

Type of model used

We used logistic regression, which was suitable because there was one dependent binary variable (Accepted vs Rejected) that was predicted by multiple nominal and binary independent variables. Logistic regression also allows us to reduce bias in the data given. Specifically, there were 400,000 data points for rejected loans with only 40,000 data points for accepted loans. This would skew our prediction model towards rejecting loans if not

counterbalanced. Using class_weight = balanced help us reduce this bias and the mean squared error of our regression model.

Training method and techniques to avoid overfitting

Besides balancing the class weights, we also used K-fold cross validation and randomized the splitting of train and test datasets. This means we are splitting the data into train and test sets randomly and tested with multiple times to avoid overfitting to one testing set. We also used L1 regularization to add a term to the cost function that imposes a penalty based on the magnitude of the parameters to reduce overfitting. This helped feature scaling because the scale of certain parameters such as loan title are larger than others such as policy codes.

Estimate of model performance on unseen data

This model is excellent at precision, recall and F1-score, which are all determinants of the accuracy and consistency of our predictions, with scores of 1.00 in each. This means that the model is very accurate in terms of determining true positives, which means predicting that loans will be accepted for those who actually got loans accepted. In terms of accuracy, this model have a score of 0.9954 which means it will very likely produce a true prediction.

The two weaknesses of this model performance is its time complexity and the type of data it is trained with. Firstly, this model is time and memory consuming because of its many data operations, for example many variables have to be turned to dummy variables and changing data types which are all computationally intensive. On my own Macbook Pro, this model actually surpassed the maximum memory usage and crashed kernel every time I ran. I had to turn to Google's Colab platform which is much more powerful to perform the algorithm. Secondly, this model was trained on data from 2007 and 2011, which meant that it could be out-of-date for the most recent data. Nevertheless, there are newer data available up until 2017 which can easily be used to train this model to add robustness and keep it updated.

Code

```
import pandas as pd
import numpy as np
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

df1 = pd.read_csv("Rejected Loans.csv",low_memory=False)
df2 = pd.read_csv("Accepted Loans.csv",low_memory=False)

dfloan = df2[['loan_amnt','title','dti','addr_state','emp_length','policy_code']]
dfloan = dfloan.dropna(how='any',axis=0) #remove NaN's

#convert the employment data into numerics
emp_map = {'emp_length':{"< 1 year":0.5,"1 year":1,"2 years":2, "3 years":3,"4 years":4,
                         "5 years":5,"6 years":6, "7 years":7,"8 years":8,"9 years":9,
                         "10 years":10,"10+ years":11}}
dfloan.replace(emp_map, inplace=True)

#adding the variable of whether they are apporved
dfloan['Approved']=pd.Series(1, dfloan.index)

dfrej = df1[['Amount Requested','Loan Title','Debt-To-Income Ratio','State',
             'Employment Length','Policy Code']]
dfrej = dfrej.rename(columns = {'Amount Requested':'loan_amnt','Loan Title':'title',
                                'Debt-To-Income Ratio':'dti','State':"addr_state",
                                'Employment Length':'emp_length',
                                'Policy Code':'policy_code'})
dfrej = dfrej.dropna() #remove NaN's
dfrej.replace(emp_map, inplace=True)
#converting dti from strings into floats
dfrej['dti']= dfrej['dti'].replace('%', '',regex=True).astype('float')
dfrej['Approved']=pd.Series(0,dfrej.index)

#joining two data sets
df = pd.concat([dfloan,dfrej])

#split predictive variables from predicted output "Approved"
dfs = np.split(df,[6],axis=1)
df = dfs[0]

#predicted outputs needs to be integers
dfy = np.ravel(dfs[1]).astype('int')

#converting categorical data into indicators
df = pd.concat([df, pd.get_dummies(df['title'], prefix = '')], axis=1)
df = pd.concat([df, pd.get_dummies(df['addr_state'], prefix = '')], axis=1)
df = df.drop('title',1)
df = df.drop('addr_state',1)

#After concatenation, convert the datatype again
df["loan_amnt"] = df["loan_amnt"].astype('int')
df["dti"] = df["dti"].astype('int')
df['emp_length'] = df['emp_length'].astype('float')
df['policy_code'] = df['policy_code'].astype('int')

#Spliting the data into training and test sets
x_train, x_test, y_train, y_test = train_test_split(df,dfy,test_size=0.35, random_state=0)
```

```
#Training the regression model
logreg = LogisticRegression(penalty='l1', class_weight='balanced', solver='liblinear',
Cs=[0.01, 0.001], refit=True, scoring='f1')
logreg.fit(x_train, y_train)
predictions = logreg.predict(x_test)

#obtain the scores to measure if it's
print metrics.accuracy_score(y_test, predictions)

#metrics for classificatioin
print metrics.r2_score(y_test, predictions)

#metrics for regression
print metrics.classification_report(y_test, predictions)

min_loan_amnt = df['loan_amnt'].min #2000

def predict_max(x):
    #input is an array of covariates that excluding loan amount
    amnt = pd.DataFrame(data = {"Approved": [1000]})
    #I picked 10000 as an arbitrary baseline because its smaller than 16000
    x = pd.concat([amnt,x],axis=1)
    for i in range(len(df.index)):
        y = logreg.predict(x)
        x.at[0,"Approved"] += 10
        if y == 0:
            max = x.at[0,"Approved"]
            break
    return max

x = df.iloc[-1]
print "The recommended maximum loan amount for an applicant with the following details is
$",predict_max(x)
print x
```