Loan Default Prediction Challenge

Xiangyu Xu

1.Introduction

As a lender, one of the core business challenges that Avant faces is predicting each potential borrower's chance of default if Avant were to approve their loan application. In this data challenge, Avant's Data Science team has prepared a small dataset that mimics the default prediction challenge that we try to solve and improve upon every day. The goal is to predict whether a borrower is going to default by learning from the pool of borrowers' behavior

2.Find Dependent Variable

Since the goal is to predict whether a borrower is going to default by learning from the pool of borrowers' behavior, I choose loan_status as the dependent variable.
There are three categories of this variable: Current, Fully Paid and Default.
Current: The borrower has paid off all due payments as of the latest due date.
Fully Paid: The borrower has paid off the entire balance of the loan.
Default: The borrower has not paid off all due payments as of the latest due date.
So I constructed a binary variable called default_status to replace loan_status, used 0 to represent the Default, and 1 to represent Current and Fully Paid, which can be seen as not default.

3.Find Predictors

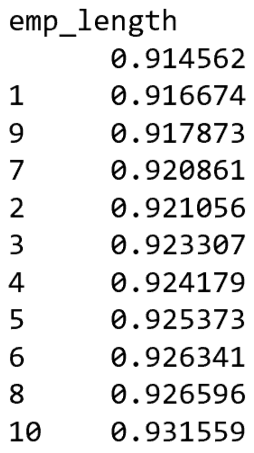
3.1 Attribute 'term'

The attribute 'term' only has two values: 60 months and 36 months.

So I convert the value 60 month into 1 and 36 month into 0, to make it a binary variable.

3.2 Attribute 'emp_length'

For the attribute 'emp_length', I notice there are some missing value n/a. I am not sure this is because these people are not employed are just missed the data during data collection.



```
emp_length
      0.914562
1     0.916674
9     0.917873
7     0.920861
2     0.921056
3     0.923307
4     0.924179
5     0.925373
6     0.926341
8     0.926596
10    0.931559
```

This picture shows the employment length and the corresponding non-default rate in an ascending order.

So I divided it into 4 groups. The people with emp_length greater than 10 have outstanding non-default rate, so itself is a group.

8,6,5 are a group; 4,3,2,7 are another one, and 9,1,missing are the last group.

Then I used one-hot encoding, which I think can reduce the burden of tree, if I am going to use tree model. Since the emp_length now has 4 unique values, so I remove one redundant feature to get three dummy variables: emp_length2, emp_length3 ,emp_length4.

3.3 Attribute 'home_ownership', 'verification_status'

The attribute home_ownership got four different values. So by the same reason, I used one hot encoding to replace this variable with three dummy variables: home_mortgage, home_own, home_rent.

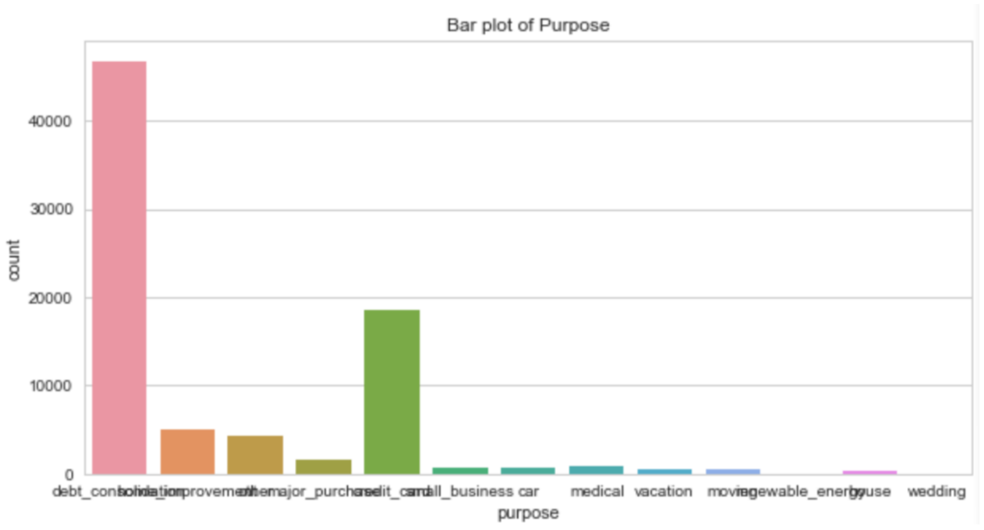
Also for the verification_status, I used one hot encoding to replace this variable with two dummy variables: income_source_verified and income_verified.

3.4 Attribute 'purpose'

For the attribute 'purpose', I also choose to one hot encode it. But this variable has 13 different category.

From the plot shows below, it is easy to see only the debt_consolidation and credit_card are outstanding.

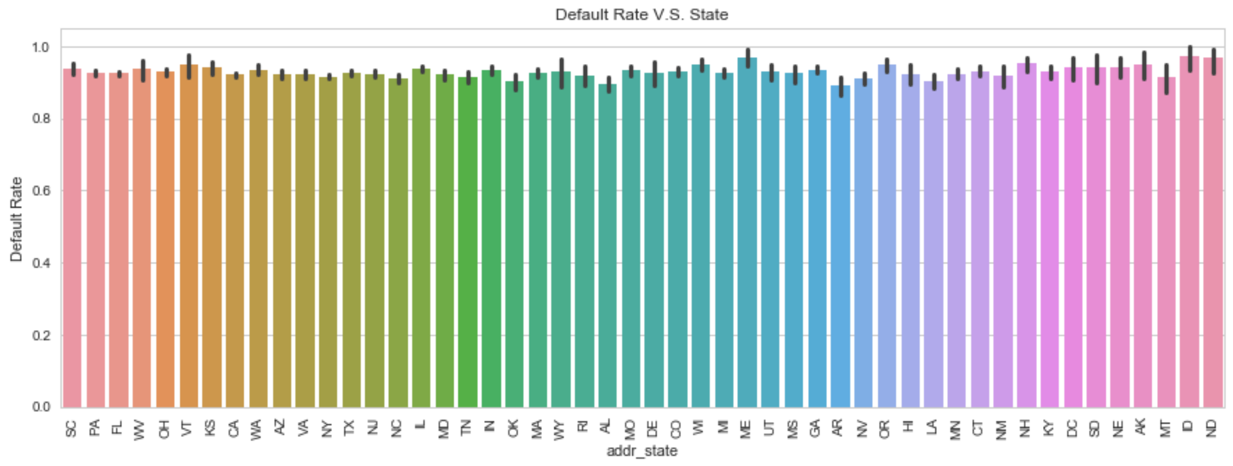
The rest variables are only small part of the total count.



So I first categorize the value into three group: debt_consolidation, credit_card and others. The group others is the sum of all the variables except the most two variables. Then one hot encode it to get two dummy variables: purpose_consolidation and purpose_creditcard.

The reason is no information loss as well as less likely overfitting.

3.5 Attribute 'addr_state'



For this attribute, I cannot one hot encode it. Because it includes 50 states, if I get 49 dummy variables, it would be too sparse.

And from the picture above although it seems that the default at the almost same level. But if we substract it from the mean value, it would be significant.

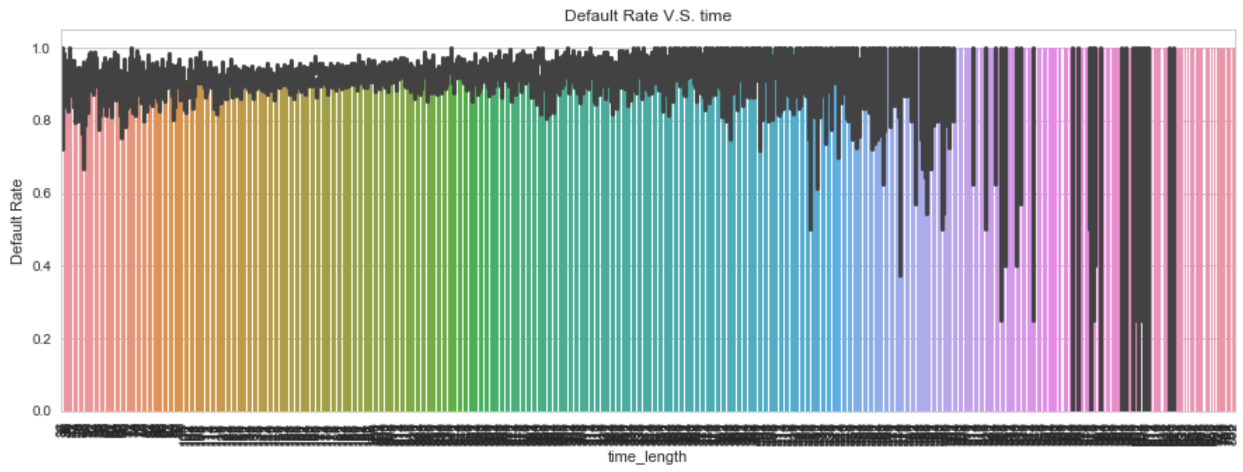
So I sort the default rate of 50 state in an ascending order, group them into 5 according to their default rate, then get four dummy variables: state_default2, state_default3, state_default4, state_default5.

3.6 Attribute ‘issue_time’ and ‘earliest_time’

Issue_time: The month which the loan was funded.

Earliest_time: The month the borrower's earliest reported credit line was opened.

These two variables cannot be directly used for analysis. So I use one to subtract another in order to get the variable time_length.



The reason is from the plot shows above, as the time_length increases, the default rate reaching 1.0 is significantly more than the customers with short time_length.

3.7 Attribute ‘fico_range_low’ and ‘fico_range_high’

First of all, I want to point out the correlation between these two variables is 1, which means one variable can linearly predict another, one of them is redundant. So I decide to calculate the mean of these two as a new feature, fico_range_mean. And I did the same for last_fico_range_high and last_fico_range_low.

3.8 Attribute ‘mths_since_last_delinq’

This variable has around 40000 missing value, but I can delete it directly, because it seems to be important predictor based on common sense and I don’t want to occur a huge information loss.

I have three solutions:

1. use KNN model to cluster the records, then use the mean of the cluster to replace the missing value.

2. use mean value to fill the NaN, and then build another (0,1) binary feature as the index. With the row whose NaN was filled by mean, the corresponding value at the binary feature would be 0.

3. use EM algorithm to help fill the missing value.

As result, I decide to use the second method, and build an index feature named mths_delinq_index.

3.9 Attribute ‘inq_last_12m’

This attribute has 42605 missing values. And based on the common sense, it has relationship with the variable inq_last_6m. Then I check for the correlation, it turns out to be 0.5 . Took all the factor into account and time constraint, I choose to delete this variable.

4.Model Building

I will choose to try logistic regression, gradient boosting, random forest as well as fully connected neural network.

But the **gradient boosting** and **random forest** would be the **best** based on the accuracy, complexity, data preprocessing concern.

The reason I need to try the linear classifier like logistic regression is because I want to see if this dataset is linearly separable. However, the its performance is highly dependent on the imputation method we choose. So even if we get high accuracy by using logistic, we still need to try others.

Gradient boosting is to fit the model on the residual left from the last tree in order to learn complex decision boundary. As the trees are getting more, we can get high accuracy. So for the dataset with many features, it would be a good choice. Also, it has an objective function to guide the model search, which is a great advantage over the other heuristic methods.

Compared with gradient boosting, random forest can build the tree parallel, which indicate it is faster than gradient boosting. But for this dataset, this advantage can be ignored. Also, it can reduce overfitting, because it is kind like bagging of decision trees.

For Neural Nets, if I can build 4 or 5 layers, I think it can also give pretty good result. But in this case, I do not think it can over power random forest and gradient boosting. Also, the time consuming is a big problem.

Based on the analysis above, I think gradient boosting and random forest would be the best.

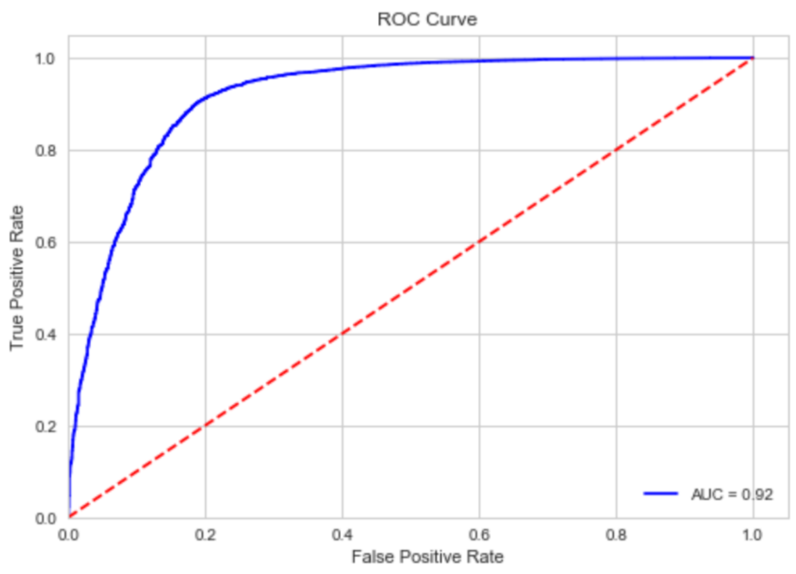
Also, I built logistic regression, gradient boosting and random forest model in Python. Because of the time constraint, I did not apply the cross-validation to find the best parameter. The reason I want put my model here is because I want to see if I get good accuracy based on the feature engineering I did. I basically split the dataset into two, 70% for training, 30% for testing.

For Logistic regression:

Accuracy of positive class: 0.900396075254

Accuracy of negative class: 0.812570145903

Roc_curve:

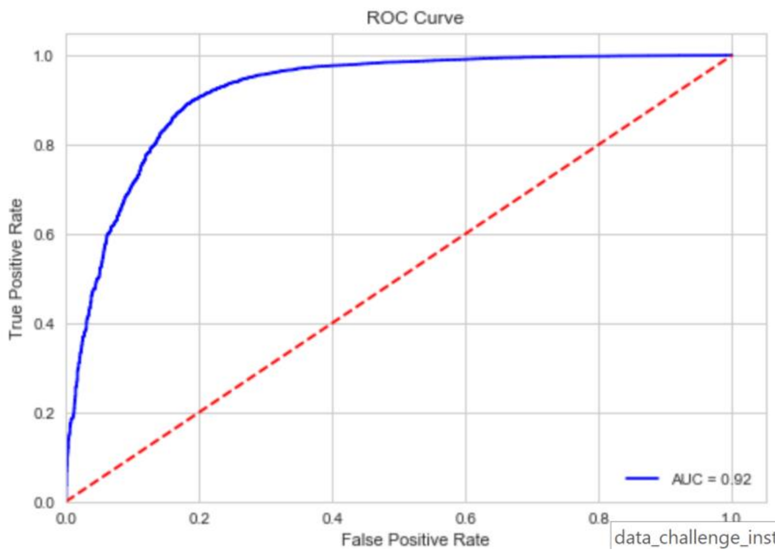


For gradient boosting:

Accuracy of positive class: 0.98199657935

Accuracy of negative class: 0.540965207632

Roc_curve



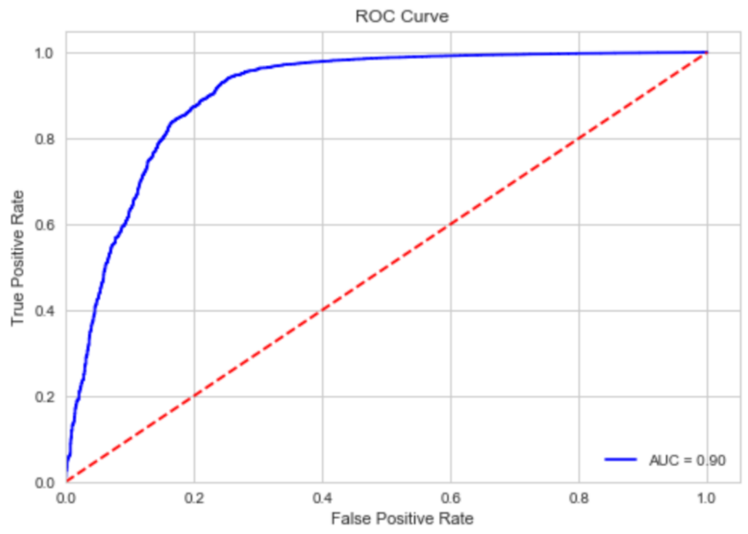
data_challenge_instru

For random forest:

Accuracy of positive class: 0.985822306238

Accuracy of negative class: 0.519640852974

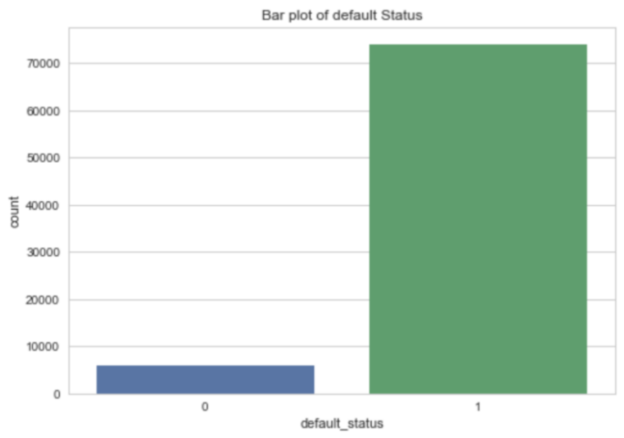
Roc_curve



5.Data Challenge

1.

For logistic regression: there is problem with overfitting and data selection bias. Since the record with default is only small part of the whole data, shows in the barplot below. So For the overfitting, I introduce the L2 norm; for selection bias, I can calculate the weight and add it to the loss function, in sklearn library, I can just use ‘balanced’ in class_weight argument, which I did in the model build part. It turns out good.



2.

For random forest,

TP :21903 FP:856 TN:926 FN:315

The false positive rate is very high due to the data selection bias. And I think the company would more care about false positive rate than others. If the customer we predict to be returning money on time actually default, it would be a loss to the company. So I will randomly select less non-default data for training every time, repeat this process for a few times, which can help get less imbalanced.

3.

For gradient boosting, the data selection bias issue also exists, and the false positive rate is also high.

So I will randomly select less non-default data for training every time, repeat this process for a few times, which can help get less imbalanced.

Also, gradient boosting prone to overfit. I would try to tune the parameter like learning rate and depth of tree.

4.

Neural nets gets easy to overfit, I would apply regularization to fix it.

6.Final Feature

'loan_amnt','term','installment','annual_inc','dti','acc_now_delinq','delinq_amnt','delinq_2yrs','mths_since_last_delinq','inq_last_6mths','purpose_car','purpose_creditcard','purpose_consolidation','home_mortgage','home_own','home_rent','fico_range_mean','last_fico_range_mean','income_source_verified','income_verified','time_length','mths_delinq_index','state_default2','state_default3','state_default4','state_default5','emp_length2','emp_length3','emp_length4'

- The variables in excel are all final features and response.

7.Model Validation

Randomly select less non-default data for training every time, repeat this process for a few times just like the k-fold cross validation. The difference is that we need to split the training and testing data separately from default and non-default data. Get the accuracy score or AUC score, then conduct the multiple hypothesis testing and Bonferroni Correction to validate model.

8.Future Work

Because of the time constraint, I did not tune the model parameter and try not many model. It also affects the data imputation method I choose.
In the future, I would like to use EM algorithm to fill up the missing data.
Second, I would try to tune the model parameter and conduct validation to find the best model.