Assignment 2: Loan Prediction

Ash

2018/12/06

Data Cleaning

Choices of variables & datasets

For both the approved data and the reject data the datasets for the first and second quarters of 2018 were chosen (LoanStats2018Q1.csv, LoanStats2018Q2.csv, RejectStats2018Q1.csv and RejectStats2018Q2.csv). This is because of three reasons:

- These are the most recent data, where the variables are more consistently coded and more coherent between different periods and also between the approve and reject data. For example, the loan title variable was coded very inconsistently in the period between 2007 and 2009: there are more than 100 different categories, many of which can conceivably be expressing the same thing. To compare, there are only 14 categories for the loan title in the Q1 and Q1 of 2018. More consistent data gives us more chance of finding the underlying model under computational constraints.

- Most financial data is a type of time-series data and is highly dependent on time. For example, stock data in 2007 cannot be used as a good proxy to model how the stock in the next month will behave, since they are separated by a long period in which many socio-economical events might have happened. Therefore if we wish to build a model that is effective in predicting near future trend we should consider more recent data. Of course for modeling long-term trend data from a more spread out period is needed.

- There are around 2.3 millions data points for both the rejected and the approved in Q1 and Q2 of 2018 alone (10% of which is approved), so we have enough data to build a reasonable model. Due to limited computational resources, we cannot use more data.

There are 6 variables chosen for analysis. Since we have way more features in the approve dataset compared to the reject datasets, the features in the reject datasets will be used to find matching features in the approve dataset. After consulting from the data dictionaries provided with the data, we selected 6 variables that are presented in both datasets by different names:

- `req_amt`: Requested amount of loan submitted to LoanClub for consideration. In the approve set there's actually another variable signifies the amount actually approved for loan, but in this specific dataset these two amounts equal, so we excluded that variable for analysis.
- `title`: The type of loan that was submitted for consideration (for example, car financing or business loan)
- `dti`: Debt-to-income ratio
- `state`: The state that the loan was submitted in
- `emp_length`: The length of employment for the person who submitted the loan, which is categorical and ranges from < 1 year to > 10 year
- `approve`: Binary, whether the loan was approved or not (1 for approved)

There are two other variables worthy of discussions. The first is `policy`, where according to the dictionary will signify if the the loan is available in public policy (1) or not (2). This potentially contains information that has predictive power but unfortunately the coding (at least for the present dataset of Q1 and Q2) is off: there is a valued 0 coded for the approved set that is not in the reject set, so if we included this feature it will be a proxy for approve/reject and will results in a very high accuracy, even though we don't know what 0 signifies, therefore we chose to drop this one. Another potentially important variable is `zip`. Especially for loan data, it feels like the zip code in the US will be a good indicator of approval because there are segregation in the zips: Black Americans, Asian Americans, Latino Americans or the aborginials usually lives in specific districts with specifc zip codes (for example, Chinatowns) and races are very much a predictive feature for loan approval since it correlates with so many other factors, especially in America, like socio-economic status or credit score. Unfortunately even when using the first 3 digits of the zip code there are 972 of them in this dataset and we don't have enough computational power to analyze everything, so we had to drop them.

Data Cleaning & Preparation

For the rejected dataset, not many features needed to be dropped. One thing to mention is that the DTI as percentage are coded as string (for example "56.7%") so we need to turn these into floats. For the same feature in the approve set it was coded as numeric but as percentages, so we also had to some pre-processing.

```
#Importing relevant libraries
% matplotlib notebook
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#Loading reject datasets, doing some data cleaning
reject_1 = pd.read_csv('/Users/ash/Downloads/LoanData/RejectStats_2018Q1.
    csv')
```

```
reject_2 = pd.read_csv('/Users/ash/Downloads/LoanData/RejectStats_2018Q2.csv')
reject = pd.concat((reject_1, reject_2), axis=0)
reject.columns = ['req_amt', 'app_date', 'title', 'risk_score', 'dti', 'zip', 'state', 'emp_length', 'policy']
reject = reject.drop(['app_date', 'risk_score'], axis=1)
reject['approve']= 0
def dumb_data_encoding_correction(x):
    return float(x.strip('%'))/100
reject['dti'] = reject['dti'].apply(dumb_data_encoding_correction)

#Loading approve datasets, doing some data cleaning
approve_1 = pd.read_csv('/Users/ash/Downloads/LoanData/LoanStats_2018Q1.csv')
approve_2 = pd.read_csv('/Users/ash/Downloads/LoanData/LoanStats_2018Q2.csv')
approve = pd.concat((approve_1, approve_2), axis=0)
approve = approve[['loan_amnt', 'title', 'dti', 'zip_code', 'addr_state', 'emp_length', 'policy_code']]
approve.columns = ['req_amt', 'title', 'dti', 'zip', 'state', 'emp_length', 'policy']
approve['approve'] = 1
def dumb_data_encoding_correction_2(x):
    return float(x)/100
approve['dti'] = approve['dti'].apply(dumb_data_encoding_correction_2)
```

After loading the data, we check for missing values and impute missing data by the most frequent values (for categorical data) and the median values (for numeric data):

```
#Check for missing data
reject.isnull().sum()
approve.isnull().sum()

#Imputing
reject['emp_length'] = reject['emp_length'].replace(np.nan, reject['emp_length'].mode()[0])
reject.isnull().sum()

approve['dti'] = approve['dti'].replace(np.nan, approve['dti'].median())
approve['emp_length'] = approve['emp_length'].replace(np.nan, approve['emp_length'].mode()[0])
approve.isnull().sum()
```

Forturnately we do not have many missing data:

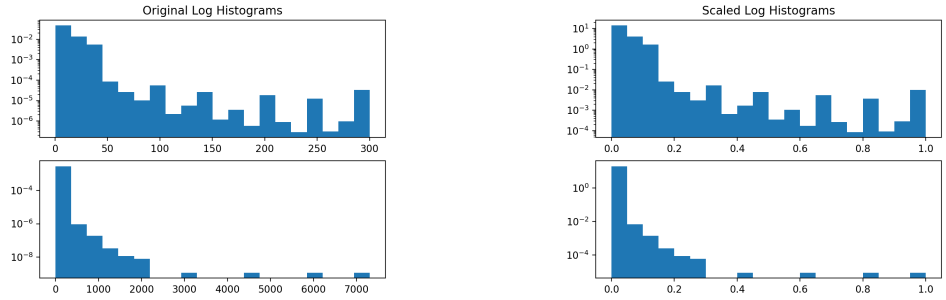
```
#Approve-before imputing
req_amt           0
title             0
dti             588
zip               0
state             0
emp_length    19896
policy            0
approve           0
dtype: int64
```

```
#Approve-after imputing
req_amt        0
title          0
dti            0
zip            0
state          0
emp_length     0
policy         0
approve        0
dtype: int64

#Reject-before imputing
req_amt            0
title              0
dti                0
zip                0
state              0
emp_length     29183
policy             0
approve            0
dtype: int64

#Reject-after imputing
req_amt        0
title          0
dti            0
zip            0
state          0
emp_length     0
policy         0
approve        0
dtype: int64
```

Combining the data:

```
loan_2018 = pd.concat((approve, reject), axis=0)
loan_2018.info()

--> Output:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2335784 entries, 0 to 1048573
Data columns (total 8 columns):
req_amt        float64
title          object
dti            float64
zip            object
state          object
emp_length     object
policy         int64
approve        int64
dtypes: float64(2), int64(2), object(4)
memory usage: 160.4+ MB
```

Since `req_amt` and `dti` are numeric we need to scale them to make sure we have



(a) Before Scaling `req_amt`-above and `dti`-below (b) After Scaling `req_amt`-above and `dti`-below

Figure 1: MinMaxScaler

meaningful coefficients. By looking at their distributions before scaling, we can see that we cannot use StandardScaler because we need to assume they are normally distributed. Using MinMaxScaler, we preserved their original distributions but scaled them to range $(0, 1)$, as in figure 1.

```
#Plot logs of dti and req_amt before scaling
plt.subplot(2,1,1)
plt.hist(np.array(loan_2018['req_amt'])/1000, bins=20, density=True, log=
    True)
plt.title('Original Log Histograms')
plt.subplot(2,1,2)
plt.hist(np.array(loan_2018['dti']), bins=20, density=True, log=True)
plt.show()

#scaling
from sklearn.preprocessing import MinMaxScaler, RobustScaler

scaler = MinMaxScaler()

loan_2018['dti'] = scaler.fit_transform(np.array(loan_2018['dti']).reshape
    (-1,1))
loan_2018['req_amt'] = scaler.fit_transform(np.array(loan_2018['req_amt']).
    reshape(-1,1))

#Plot logs of dti and req_amt after scaling
plt.subplot(2,1,1)
plt.hist(np.array(loan_2018['req_amt']), bins=20, density=True, log=True)
plt.title('Scaled Log Histograms')
plt.subplot(2,1,2)
plt.hist(np.array(loan_2018['dti']), bins=20, density=True, log=True)
plt.show()
```

We also use the one-hot-encoding scheme for categorical data:

```
#One-hot-encoding categorical data
title = pd.get_dummies(loan_2018['title'], prefix='ti')
state = pd.get_dummies(loan_2018['state'], prefix='st')
employ = pd.get_dummies(loan_2018['emp_length'], prefix='em')
```

```
5
6 loan_2018 = loan_2018.drop(['title', 'state', 'emp_length', 'policy'], axis
    =1)
7 loan_2018 = pd.concat((loan_2018, title, state, employ), axis=1)
8
9 #Dropping zip, save processed data
10 loan_2018 = loan_2018.drop(['zip'], axis=1)
11 loan_2018.to_csv('/Users/Ash/Downloads/LoanData/loan_2018_0zip.csv')
12
13 #Spliting to training and testing
14 from sklearn.model_selection import train_test_split
15
16 Train, Test, _, _ = train_test_split(loan_2018, loan_2018, test_size=0.2,
    random_state=42)
```

Modeling

Since we are dealing with a problem without groundtruths, we need to be creative about the modeling process. We need to find the maximum amount of loan that a person with a specific set of values for the above 6 features should apply for and get approval, but we don't have the maximum amounts for any training data to train a regressor on (since for all we know the approved people can applied for 10 dollars and got approved, but they could have applied for 1000 and still got approved). First, we need a good measure of how any of our modeling will work, we will train a baseline classifier that takes in 5 features (including the requested amount, but excluding the approval status) and tell us whether the person will get approved or not. If we have a decent classifier, we can use it to test whether the predicted maximum amount of requested loan will be approved or not.

```
1 from sklearn.linear_model import LinearRegression, LogisticRegression
2 from sklearn.ensemble import RandomForestClassifier
3 from sklearn.model_selection import cross_val_score
4 from sklearn.metrics import confusion_matrix
5
6 '''
7 Baseline Classification Model
8 '''
9
10 #Using LogisticRegression
11 classifier = LogisticRegression(penalty='l2', solver='lbfgs')
12
13 #Using approval status as label
14 clf_label = Train['approve']
15 clf_feature = Train.drop(['approve'], axis=1)
16
17 #Cross-validation on the training set
18 acc = cross_val_score(classifier, clf_feature, clf_label, cv=5, scoring='
    accuracy')
19 f1 = cross_val_score(classifier, clf_feature, clf_label, cv=5, scoring='f1'
    )
```

```
auc = cross_val_score(classifier, clf_feature, clf_label, cv=5, scoring='roc_auc')

print('CV acc mean:', np.mean(acc))
print('CV f1 mean:', np.mean(f1))
print('CV auc mean:', np.mean(auc))

#Saving the classifier for future usages
classifier.fit(clf_feature, clf_label)
import pickle

with open('/Users/ash/Downloads/base_classifer.model', 'wb') as f:
    pickle.dump(classifier, f)

-->Output:
CV acc mean: 0.9689772223540913
CV f1 mean: 0.8581259727974723
CV auc mean: 0.9638433457027024
```

Using a logistic regression with l-2 regularization, we can see that the classifier achieves very good results in cross-validation. We now have a baseline model that is very good at differentiating the approved and the rejected, taking into account the amount requested.

Model Idea 1: Conservative Prediction

The first idea is to use a simple linear regression on the approve set only, using the requested amount as the groundtruths. In this idea we are assuming that with the people who were approved, the approved amount is the maximum amount they can get (which is not necessarily true), but if we use this regressor to predict for the reject set we will hopefully get a conservative guess of the amount they should apply for to get approval.

```
'''
Idea 1: A conservative guess for rejected individuals
'''
#Separating into approve and reject set
approve = Train.loc[Train['approve'] == 1]
reject = Train.loc[Train['approve'] == 0]
print('Number of approve:', approve.shape[0])
print('Number of reject:', reject.shape[0])

#Simple LinearRegression to predict the amount of requested (and approved) for the approve set
approve_label = approve['req_amt']
approve_feature = approve.drop(['req_amt', 'approve'], axis=1)

model_maxfund = LinearRegression()
mse = cross_val_score(model_maxfund, approve_feature, approve_label, cv=2, scoring='neg_mean_squared_error')
mae = cross_val_score(model_maxfund, approve_feature, approve_label, cv=2, scoring='neg_mean_absolute_error')
print('CV MSE Mean:',-np.mean(mse))
```

```
print('CV MAE Mean: ',-np.mean(mae))

-->Output:
CV MSE Mean: 0.0010976587403326967
CV MAE Mean: 0.027006318289891286
```

This seems pretty good, but we are predicting on the scaled units. Let's go back to the original unit:

```
model_maxfund.fit(approve_feature, approve_label)
preds = model_maxfund.predict(approve_feature)
preds = scaler.inverse_transform(preds.reshape(-1,1))
truths = scaler.inverse_transform(np.array(approve_label).reshape(-1,1))
print('MSE in original units:', np.mean((truths-preds)**2))
print('MAE in original units:' ,np.mean(np.abs(truths-preds)))

-->Output:
MSE in original units: 98438644.53607078
MAE in original units: 8087.041402282708
```

So we are off by around 8000 dollars on average. Not a super good result, but to be expected for a very complex problem of loan prediction with only 5 features to work with. Linear regression does not work so well (even without regularization) suggesting that the problem is very non-linear (in reality, testing with non-linear regressors, like random forest, returns a better results: MAE around 5000 dollars). A small test to see if the regressor is making sense is to try predicting for the reject set: if the regressor learned some useful relationship then the predicted maximum amount of requested for the reject should be less than theirs original requested amount.

```
reject_label = reject['req_amt']
reject_feature = reject.drop(['req_amt', 'approve'], axis=1)

reject_maxfund = model_maxfund.predict(reject_feature)
sum(reject_label > reject_maxfund)/reject_label.shape[0]

-->Output: 0.30460000655653674
```

Again, not very good results, but we can see that 30% of the rejected actually was recommended by the regressor to lower their requested amount, meaning that the regressor actually learned something useful (testing with the mentioned random forest model yielded around 33%).

Model Idea 2: Conservative Prediction - Logistics Probability

The second idea is to use the baseline model, but since we used LogisticRegression we have a raw probabilities output (for example, a particular applicant has 80% chance of getting approval). Using these probabilities, we again has a conservative guess of the amount someone should requested: for the approve the probability is ideally really near 1, so their maximum amount stay approximately the same when multiplying the requested amount the probabilities, but for the rejected they has low probabilities of

approval, so multiplying these probabilities with the amount they originally requested will give us an idea how much they probably should have applied for to get approval.

```
'''
Idea 2: Probability Thresholding
'''
from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(penalty='l2', solver='lbfgs')

proba_label = Train['approve']
proba_feature = Train.drop(['approve'], axis=1)

classifier.fit(proba_feature, proba_label)
probas = classifier.predict_proba(proba_feature)

maxfund = np.multiply(scaler.inverse_transform(np.array(proba_label).
    reshape(-1,1)),probas[:,1])
```

Model Idea 3: Clustering & Regression

The third idea is do a semi-supervised learning. First, we cluster the data with KMeans clustering (with $K = 10$, there are room for fine-tuning here). Then, for each cluster, we calculate the median of the cluster requested amount, then assigned each person the maximum amount of approval according to their status: if the person was originally approved, take the max(its cluster median, its requested amount) and if the person was originally rejected, take the min(its cluster median, its requested amount). These are then used to train as groundtruths for a regressor.

```
'''
Idea 1: K-Means Clustering
'''
#Using a slice of the data due to computational constraints
mini_train, _, _, _ = train_test_split(Train, Train, train_size=0.2)

from sklearn.cluster import KMeans, MiniBatchKMeans

mini_loan = mini_train['req_amt']
mini_approve = mini_train['approve']
mini_f = mini_train.drop(['req_amt', 'approve'], axis=1)

#10 clusters from KMeans
cluster = KMeans(n_clusters=10)
preds = cluster.fit_predict(mini_f)

#Calculating the medians of each cluster
mini_train['cluster'] = preds
cluster_median = mini_train.groupby(mini_train['cluster'])['req_amt'].
    median()

#Assigning each person either the min or the max
l = []
```

```
for _ in range(mini_train.shape[0]):
    if mini_train.iloc[_,2] == 0:
        l.append(min(cluster_median[mini_train.iloc[_,-1]],mini_train.iloc[_,0]))
    elif  mini_train.iloc[_,2] == 1:
        l.append(max(cluster_median[mini_train.iloc[_,-1]],mini_train.iloc[_,0]))
mini_train['predicted_max_amt'] = np.array(l)

#Training a simple regressor using the clustering groundtruths
mini_label = mini_train['predicted_max_amt']
mini_feature = mini_train.drop(['predicted_max_amt', 'cluster', 'req_amt', 'approve'],axis=1)
reg = LinearRegression()

reg.fit(mini_feature, mini_label)
preds = reg.predict(mini_feature)
preds = scaler.inverse_transform(preds.reshape(-1,1))
truths = scaler.inverse_transform(np.array(mini_label).reshape(-1,1))
print('MSE in original units:', np.mean((truths-preds)**2))
print('MAE in original units:' ,np.mean(np.abs(truths-preds)))

-->Output:
MSE in original units: 15721813.80325526
MAE in original units: 2762.4051739792653
```

It achieves quite good MAE, but since we don't really have groundtruths this is just a suggestive result.

Prediction

It's time to use the test set to test our model performance. First, we use the original baseline classifier to predict on the test set to see if it's performing as good as we expected.

```
'''
Testing Performance
'''
from sklearn.metrics import roc_auc_score, accuracy_score, f1_score
test_label = Test['approve']
test_feature = Test.drop(['approve'], axis=1)

test_preds = classifier.predict(test_feature)
print('Test Acc:', accuracy_score(test_label, test_preds))
print('Test F1:', f1_score(test_label, test_preds))
print('Test AUC of ROC:', roc_auc_score(test_label, test_preds))

-->Output:
Test Acc: 0.9690382462426979
Test F1: 0.8586782350412319
Test AUC of ROC: 0.9476424731933075
```

It achieves similar performances with the test set compared to the training set on the calculated metrics. It has nice balance of precision and recall (by the F1 score, which is the harmonic mean of precision and recall), and also nice TPR and FPR tradeoff as expressed by the AUC of the ROC. We are pretty confidence that this will generalize well with unseen data. Now, using the three ideas presented in the previous section, let's see if we are able to recommend the rejected a more reasonable maximum amount they ought to request.

Model Idea 1: Conservative Prediction

```
test_approve = Test.loc[Test['approve'] == 1]
test_reject = Test.loc[Test['approve'] == 0]

# Test Idea 1:
#Using the model_maxfund to predict for the test reject set
test_reject_1 = test_reject.drop(['approve'], axis=1)
test_reject_feature = test_reject_1.drop(['req_amt'], axis=1)
predict_1 = model_maxfund.predict(test_reject_feature)
test_reject_1['req_amt'] = predict_1

#Using the baseline classifier to predict the approval status with new
    predicted req_amt
confirm_1 = classifier.predict(test_reject_1)
print('Total Approved For Predicted Max Fund, Model 1:', sum(confirm_1)/len
    (confirm_1))

-->Output:
Total Approved For Predicted Max Fund, Model 1: 0.025257567584482464
(Originally, without predicted max fund by model_maxfund:
    0.025467714748243463)
```

There seems to be even a decrease in the predicted number of people being rejected. It seems that idea 1 is not very good at predicting the conservative maximum amount. Using the maxfund model to predict for the approve though, it achieves decent results with about 91% of the originally approved people get re-approved with the new amount of requested loan.

Model Idea 2: Conservative Prediction - Logistics Probability

```
# Test Idea 2:
test_reject_2 = test_reject.drop(['approve'], axis=1)
probas = classifier.predict_proba(test_reject_2)
maxfund = np.multiply(probas[:,1], test_reject_2['req_amt'])
test_reject_2['req_amt'] = maxfund

confirm_2 = classifier.predict(test_reject_2)
print('Total Approved For Predicted Max Fund, Model 2:', sum(confirm_2)/len
    (confirm_2))

-->Output:
Total Approved For Predicted Max Fund, Model 2: 0.025458178064196188
```

```
(Originally, without predicted max fund by thresholding probabilities:
    0.025467714748243463)
```

Again, there is a very slight decrease in the predicted approval status for the newly predicted maximum request amount. It is also not a very good model, one which believe will generalize well with new data.

Model Idea 3: Clustering & Regression

```
# Test Idea 3:
test_reject_3 = test_reject.drop(['approve'], axis=1)
test_reject_feature_3 = test_reject_3.drop(['req_amt'], axis=1)
clusters = cluster.predict(test_reject_feature_3)
test_reject_3['cluster'] = clusters

l = []
for _ in range(test_reject_3.shape[0]):
    if test_reject_3.iloc[_,2] == 0:
        l.append(min(cluster_median[test_reject_3.iloc[_,-1]],test_reject_3
    .iloc[_,0]))
    elif  test_reject_3.iloc[_,2] == 1:
        l.append(max(cluster_median[test_reject_3.iloc[_,-1]],test_reject_3
    .iloc[_,0]))

test_reject_3['req_amt'] = np.array(l)
test_reject_3 = test_reject_3.drop(['cluster'], axis=1)
confirm_3 = classifier.predict(test_reject_3)
print('Total Approved For Predicted Max Fund, Model 3:', sum(confirm_3)/len
    (confirm_3))

-->Output:
Total Approved For Predicted Max Fund, Model 3: 0.025259951779652682
(Originally, without predicted max fund by thresholding probabilities:
    0.025467714748243463)
```

So none of the models can be generalized well to unseen data. :(

Discussion

This is a very complicated task, in which we have to perform learning without groundtruths. Three models presented above are among many others that can be devised. Unfortunately, none from above generalizes well with unseen data, probably because many factors. First, the original baseline classification model coefficients is quite strange. DTI coefficient (at the second place in figure 2) is negative, which is understandable since the more DTI you have the less likely you will get a loan, and the last coefficient (for employment $<$ 1 year) is very negative suggesting that if you have been employed for less than 1 year it's very unlikely that you will have a chance of getting a loan. These all seem reasonable, but the first coefficient representing requested amount is positive, suggesting that the more you requested the more likely you will get approved, which is quite counterintuitive. Looking back at the data distributions for the requested amount of the two groups in this specific set (Q1+Q2), it is indeed the case that the approve

group generally requested more than the rejected group (abeit there are some outliers in the rejected group who requested up to 300,000 dollars). This is interesting, and it would be beneficial to look more at the historical data to see if it's the case. Secondly, there are much room for fine-tuning and testing with different models for each of the direction of the three ideas above. For example, instead of using linear regression we can use a more sophisticated methods that can detect non-linearity in the data such as SVM or random forest, or even ensemble models.

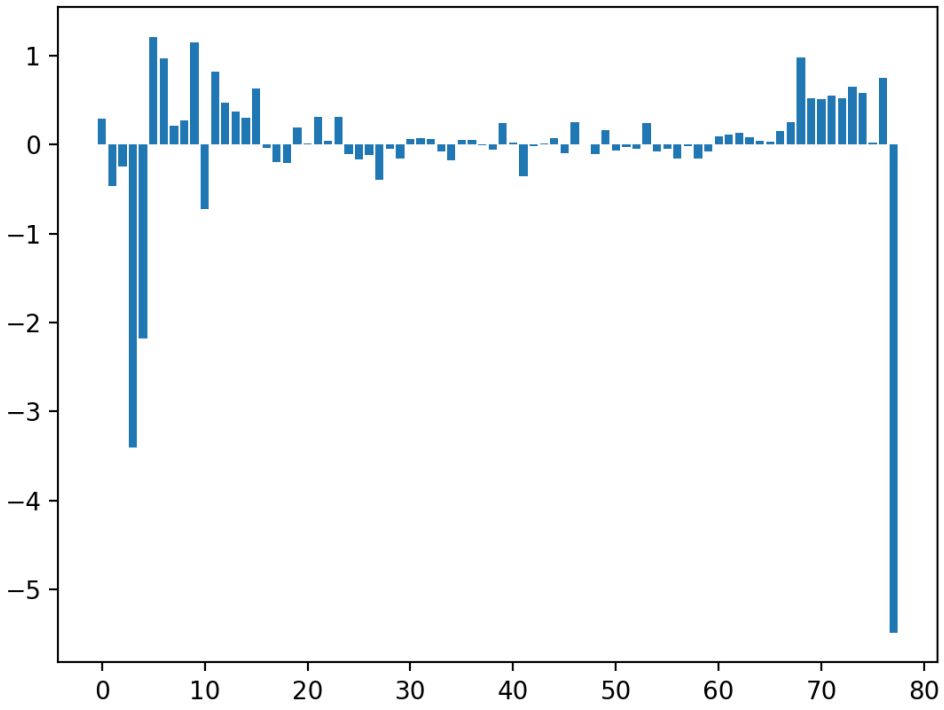


Figure 2: Coefficients of the classification models - Bar Plot

The direction from here might be actually considering the classification model, which performs really well with unseen data. With this model, we will get a probability of getting approved, which we can interpret as the amount of risk the LoanClub is taking when loaning out money. We then can incorporate these risks with a realistic utility model and set a cutoff financially utility for LoanClub, then calculate the maximum amount it willing to loan out for a requesting individual with a specific predicted probabilities based on this utility model. In this case, using a non-linear utility model that reflects aversion to loss, such as prospect theory of Kahneman and Tversky, might be a good start.