Roujia Wen
CS156
February 7, 2018

Assignment 2 - Lending Club Data

Cleaning Data

Raw Data Selection. I included 16 data files downloaded from lendingclub.com – data for both approved and rejected requests from 2015 to the third quarter of 2017 (latest).

Feature Selection. There were eight features in common between the approved and the rejected files. Among them, "Policy Code" was completely correlated with the outcome (whether the request was approved or rejected) and therefore dropped[1]; "Zip Code" was also excluded for model simplicity and limitations in computational performance. Variables which remained for further analysis include: Amount Requested, Application Date[2], Loan Title, Debt-To-Income Ratio (DTI), State and Employment Length.

Missing Data. Since observations with missing features were only a small proportion of the data, I decided to remove them. Observations with a DTI value of -1% were treated as missing/spurious data[3]. As a result, 6.43% were dropped in the approved and 10.88% were dropped in the rejected data.

Relabeling Loan Titles. In Loan Title, 14 unique string values (ignoring cases) were common in both approved and rejected data files while 23 appeared only in rejected data. I relabelled the latter to one of the 14 major categories according to mapping_dict. For example, loan —> debt_consolidation; car financing —> car; medical expenses —> medical.

Reformatting. converted DTI in the rejected data from string to float, and created a new column with Employment Length as integers (originally categorical) for exploratory purposes.

Combining. Since there are 7.5e5 approved observations and 1.13e7 rejected ones, I resampled the rejected data so that it has the same size as the approved data, and concatenated them. An Outcome column was added to the combined data, indicating whether the request was approved (1) or rejected (0).

Categorical variables. I used one-hot-encoding (adding k-1 dummy variables for k categories) for Employment Length, State and Title.

[1]The data dictionary did not make it clear what Policy Code meant. However, it's reasonable to believe that it is ex-post and therefore not useful to include for this analysis.

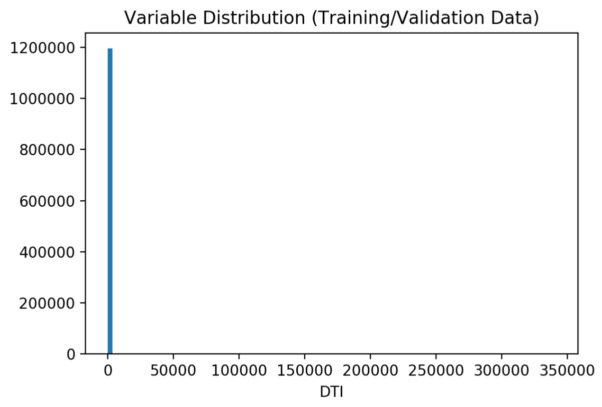
[2] Application Date did not end up being used at the end.

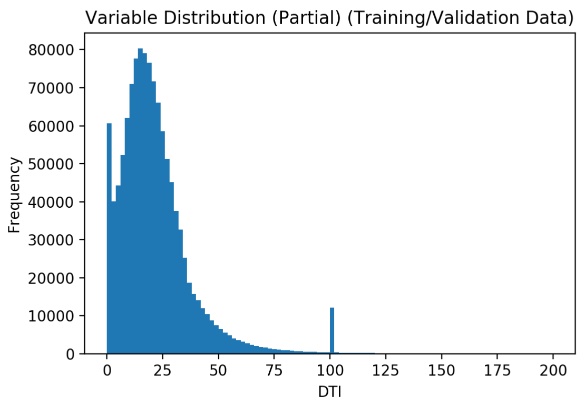
[3] Negative debt-to-income ratio does not make sense. In addition, the only negative DTI in the data were -1%, which indicates that this is an artifact of logistics rather than actual DTI.

Splitting Data

So far, no decision has been made basing on variable distribution or model performance, in order to prevent information leakage. Thus in this step, I split the data at an 8:2 ratio into training/validation data, and testing data. I hided all the testing data until the model is completely finalized, and only used them to show the performance of the final model. From now on until [] section, “data” only refers to the training/validation data.

Examining Distributions





From the graphs above we see that DTI has an extremely long-tailed distribution. When we zoom in to the range between 0 and 200, we see that it looks like a partial bell-shaped curve except for high peaks at 0 and 100. I suspect that the peak at 100 is because some people enter 100% when the actual DTI exceeds 100% due to some sort of logistic/psychological reasons.

I suspect that there is very little chance for a person with extremely large DTI to get a loan. Indeed, only 0.097% of those with DTI>100% gets approved[4]. Therefore I decided to not include these data for model training, since accounting for those extreme values might compromise the model performance for the more practically useful domain (DTI at 0-100%)[5].

All other data transformations are specified in the table on the next page.

I included logged amount since it has been observed[6] that many economical variables provide more useful information when log-transformed. A quadratic term was also included for exploratory purposes. In addition, the rational for using Robust Scaler is that it is less sensitive to extreme values, which strongly bias the Min-Max Scaler.

[4] This number is taken from all non-missing data in 2015-2017Q3.

[5] When a test datapoint has DTI > 100%, we could either just predict it to be rejected, or build a separate model (if it’s worth the effort).

[6] Lütkepohl, H., & Xu, F. (2012). The role of the log transformation in forecasting economic variables. *Empirical Economics*, *42*(3), 619-638.

	DTI	Amount Requested	Logged Amount	Squared Amount
Original Distribution	≈Bell-shaped	Long-tailed	≈Bell-shaped	Long-tailed
Scaler Applied	MinMax	Robust	MinMax	Robust
Resulted Distribution Statistics	mean 0.215 std 0.163 min 0.000 25% 0.110 50% 0.186 75% 0.278 max 1.000	mean 0.237 std 0.668 min -0.633 25% -0.333 50% 0.000 75% 0.667 max 10.443	mean 0.511 std 0.155 min 0.000 25% 0.396 50% 0.516 75% 0.635 max 1.000	mean 0.491 std 1.011 min -0.266 25% -0.200 50% 0.000 75% 0.800 max 73.793

Table - Data distributions, transformations and and resulted statistics.

Logistic Regression without Interaction Terms

While hand-tuning the model and deciding which variables to include, I've made the following observations and decisions (5-fold cross validation was used to calculate prediction accuracy):

- The model performs significantly better using Employment Length as categorical variables rather than a continuous variable. Improvement in accuracy was roughly 10.9%. This makes sense because there are a lot more assumptions made when treating it as a continuous variable such as monotonicity and linearity in its effect on the outcome.
- Amount Requested alone can predict outcome with 56.4% accuracy. Amount + Log Amount together can achieve 61.5%. However, together with the other predictors, whether to use Amount or/and Log Amount do not make a significant difference. Therefore I decided to simply use Amount for model simplicity and easiness of interpretation.
- State does not make a significant contribution to model performance and is therefore discarded.
- The most parsimonious best-performing model is trained with Amount, DTI, Loan Title and Employment Length (categorical). The validation accuracy is 89.7%.
- However, taking Amount out of the model, the accuracy is only 0.1% lower. In addition, the regression coefficient for Amount is always positive.

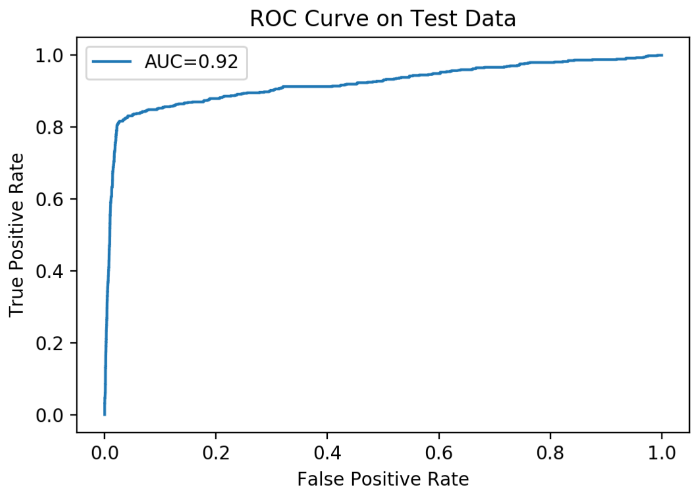
To summarize, though achieving a fairly good validation accuracy, I failed to address the question "What's the largest amount of loan that will be successfully funded for given individual?" since the variable Amount is almost irrelevant in this classifier and what's worse is that the model simply tells us that the more you request, the more likely you'll get an approval.

Logistic Regression with Interaction Terms

What I thought was missing from this very linear and simplistic model are some interaction terms. For example, the combination of certain Employment Length and Amount Requested might have other effects than simply the superposition of their separate effects. Therefore, I included Amount $\times$ Employment Length and Amount $\times$ DTI in the final model.

Evaluation

Performing the exact same data transformation on the test set[7], the model yielded a surprisingly high prediction accuracy at 94.7% on these unseen data. An ROC Curve was plotted and the area under curve was high as 0.92.

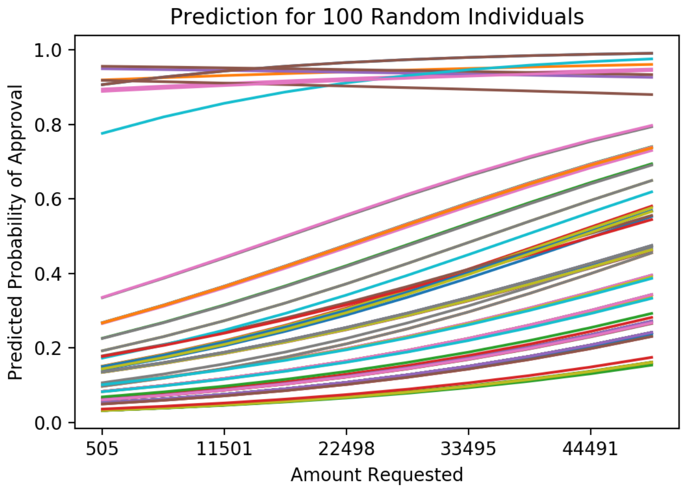


This shows that the model has a very good performance in predicting whether certain request would get approved or rejected. However, when I tried to "reverse" the process by plotting the probability of a request[8] getting approved as a function of loan amount, I got some interesting results (see figure on the next page).

First, 96.68% of the time the probability increases as the loan amount increases. As we can see from the plot, the only descending lines are almost all located at the top, which seems to say that those individuals have really good profiles so that they get approved no matter how much they ask.

[7] I transformed the test data with the scalers that were originally fitted to the training data to avoid information leakage.

[8] Each "hypothetical" request was made up of an actual individual from the test set except for the amount requested (which varies).



My explanation for the overall trend that "the more you request, the more likely you get approved" is that there is the perfect doctor effect in the process of assigning "treatment." That is, individuals are already adjusting their requested loan amount basing on what is expected as being reasonable. The people with better credit profile tend to request more and those with poorer profile request less, which leads to correlation between the Amount variable and the other independent variables. This makes it very hard for the model to extrapolate, for example, to predict what would happen if someone with a poor credit profile requests a huge amount of loan. I hypothesize that this explains why a lot of the curves at the bottom are ascending - it's simply erroneous extrapolation due to the lack of knowledge.

To conclude, the logistic regression model does a good job predicting whether a give loan request would get approved. However, it extrapolates poorly in spaces where observations are sparse, and cannot be used to address the question of the appropriate amount for a request.

Assignment 2

February 8, 2018

1 Preprocessing

1.1 Import and concatenate

```
In [1]: import pandas as pd
        app_dfs = []
        rej_dfs = []

        for year in ["2015","2016", "2017"]:
            for quarter in ["", "Q1", "Q2", "Q3", "Q4"]:
                try:
                    app_dfs.append(pd.read_csv("LoanStats_"+year+quarter+".csv", header=1, low_m
                    rej_dfs.append(pd.read_csv("RejectStats_"+year+quarter+".csv", header=1, low
                except:
                    pass
```

```
In [2]: # Verify all 16 files are included
        print len(app_dfs), len(rej_dfs)
```

```
8 8
```

```
In [3]: # Verify all files have the same column names
        import numpy as np
        for i in range(len(app_dfs)):
            print np.all(app_dfs[0].columns.values == app_dfs[i].columns.values),
        for i in range(len(rej_dfs)):
            print np.all(rej_dfs[0].columns.values == rej_dfs[i].columns.values),
```

```
True True True True True True True True True True True True True True True True
```

```
In [4]: # The last two rows in each approved file are not actual data
        for i in range(len(app_dfs)):
            print app_dfs[i].iloc[-2:, 0]
```

```
42536    Total amount funded in policy code 1: 460296150
42537            Total amount funded in policy code 2: 0
```

```
Name: id, dtype: object
133887    Total amount funded in policy code 1: 2087217200
133888     Total amount funded in policy code 2: 662815446
Name: id, dtype: object
97854    Total amount funded in policy code 1: 1443412975
97855     Total amount funded in policy code 2: 511988838
Name: id, dtype: object
99120    Total amount funded in policy code 1: 1404586950
99121     Total amount funded in policy code 2: 567447023
Name: id, dtype: object
103546    Total amount funded in policy code 1: 1465324575
103547     Total amount funded in policy code 2: 521953170
Name: id, dtype: object
96779    Total amount funded in policy code 1: 1437969475
96780     Total amount funded in policy code 2: 520780182
Name: id, dtype: object
105451    Total amount funded in policy code 1: 1538432075
105452     Total amount funded in policy code 2: 608903141
Name: id, dtype: object
122701    Total amount funded in policy code 1: 1791201400
122702     Total amount funded in policy code 2: 651669342
Name: id, dtype: object
```

```
In [5]: # Therefore getting rid of the last two rows
        for i in range(len(app_dfs)):
            app_dfs[i] = app_dfs[i].iloc[:-2]
```

```
In [6]: app_allcol = pd.concat(app_dfs).reset_index(drop=True)
        rej_allcol = pd.concat(rej_dfs).reset_index(drop=True)
```

1.2 Select relevant columns and handle missing/spurious data

```
In [7]: app_relevant_columns = ["loan_amnt",
                                "issue_d",
                                "purpose",
                                "dti",
                                "addr_state",
                                "emp_length"]
                                #"policy_code"]
        rej_relevant_columns = ['Amount Requested',
                                'Application Date',
                                'Loan Title',
                                'Debt-To-Income Ratio',
                                'State',
                                'Employment Length']
                                #'Policy Code']
        app_st = app_allcol[app_relevant_columns].copy()
        rej_st = rej_allcol[rej_relevant_columns].copy()
```

```
In [8]: # Standardize column names
        column_names = ["amount", "date", "title",  "dti", "state", "emp_length"]
        rej_st.columns = app_st.columns = column_names
```

```
In [9]: # dti == -1% does not make sense and is considered missing/spurious data
        app_st["dti"].replace(-1.,np.NaN, inplace=True)
        rej_st["dti"].replace("-1%",np.NaN, inplace=True)

        # Overview of missing data situation
        print "Percentage of missing data for each column"
        for df in [app_st, rej_st]:
            print
            print (len(df)-df.count())/float(len(df))*100
```

```
Percentage of missing data for each column

amount        0.000125
date          0.000125
title         0.000125
dti           0.044645
state         0.000125
emp_length    6.419462
dtype: float64

amount        0.000000
date          0.000000
title         0.010165
dti           6.632155
state         0.000008
emp_length    4.541271
dtype: float64
```

```
In [10]: # Given that missing data is a small fraction, I decided to drop any row that contains
         app_st = app_st.dropna()
         rej_st = rej_st.dropna()
         print "new vs. original df size ratio:"
         print "approved", len(app_st)/float(len(app_allcol))
         print "rejected", len(rej_st)/float(len(rej_allcol))
         print "\nfinal size:"
         print "approved", len(app_st)
         print "rejected", len(rej_st)
```

```
new vs. original df size ratio:
approved 0.935784175569
rejected 0.89114974373

final size:
approved 750381
```

```
rejected 11300972
```

1.3 standardize labels and convert formats

1.3.1 Handle labels of "title" column

```
In [11]: # Convert to all lowercase
         app_st.loc[:,"title"] = app_st["title"].str.lower()
         rej_st.loc[:,"title"] = rej_st["title"].str.lower()
```

```
/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages/pandas/core/indexi
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#
  self.obj[item] = s
```

```
In [12]: app_titles = set(app_st["title"].unique())
         rej_titles = set(rej_st["title"].unique())

In [13]: # See which labels have no counterparts
         print app_titles-rej_titles
         print rej_titles-app_titles
```

```
set(['educational'])
set(['loan', 'business advertising loan', 'car financing', 'althea9621', 'consolidate debt', 'mo
```

```
In [14]: len(app_titles)

Out[14]: 14

In [15]: len(rej_titles-app_titles)

Out[15]: 23

In [16]: mapping_dict = {
             'loan':'debt_consolidation',
             'business advertising loan':'small_business',
             'car financing':'car',
             'althea9621': 'other',
             'consolidate debt':'debt_consolidation',
             'moving and relocation':'home_improvement',
             'medical expenses':'medical',
             'thad31':'other',
             'home buying':'house',
             'freeup':'debt_consolidation',
             'small business expansion':'small_business',
             'business loan':'small_business',
             'need a decent rate on car financing':'car',
             'business':'small_business',
             'auto financing':'car',
             'major purchase': 'major_purchase',
             'business line of credit':'small_business',
             'dougie03':'other',
             'debt consolidation':'debt_consolidation',
             'credit card refinancing':'credit_card',
             'smmoore2':'other',
             'home improvement':'home_improvement',
             'green loan':'renewable_energy',
             'consolidation loan':'debt_consolidation',
             '10 months away from being an rn':'educational'
         }
         mapping_dict.update({_:_ for _ in app_titles})
```

```
In [17]: rej_st["title"] = rej_st["title"].map(mapping_dict)
```

```
/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages/ipykernel_launcher
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#
  """Entry point for launching an IPython kernel.
```

1.3.2 Convert debt-income ratio

```
In [18]: def percent_to_float(s):
             assert isinstance(s, basestring) and (s[-1] == "%")
             return float(s[:-1])
         rej_st["dti"] = rej_st["dti"].map(percent_to_float)
```

```
/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages/ipykernel_launcher
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#
  after removing the cwd from sys.path.
```

1.3.3 Map employment length to integers

```
In [112]: def emp_length_to_int(s):
              if s[0] == "<":
                  return 0
              elif s[:3] == "10+":
                  return 10
```

```
            return int(s[0])
        rej_st["emp_length_int"] = rej_st["emp_length"].map(emp_length_to_int)
        app_st["emp_length_int"] = app_st["emp_length"].map(emp_length_to_int)
```

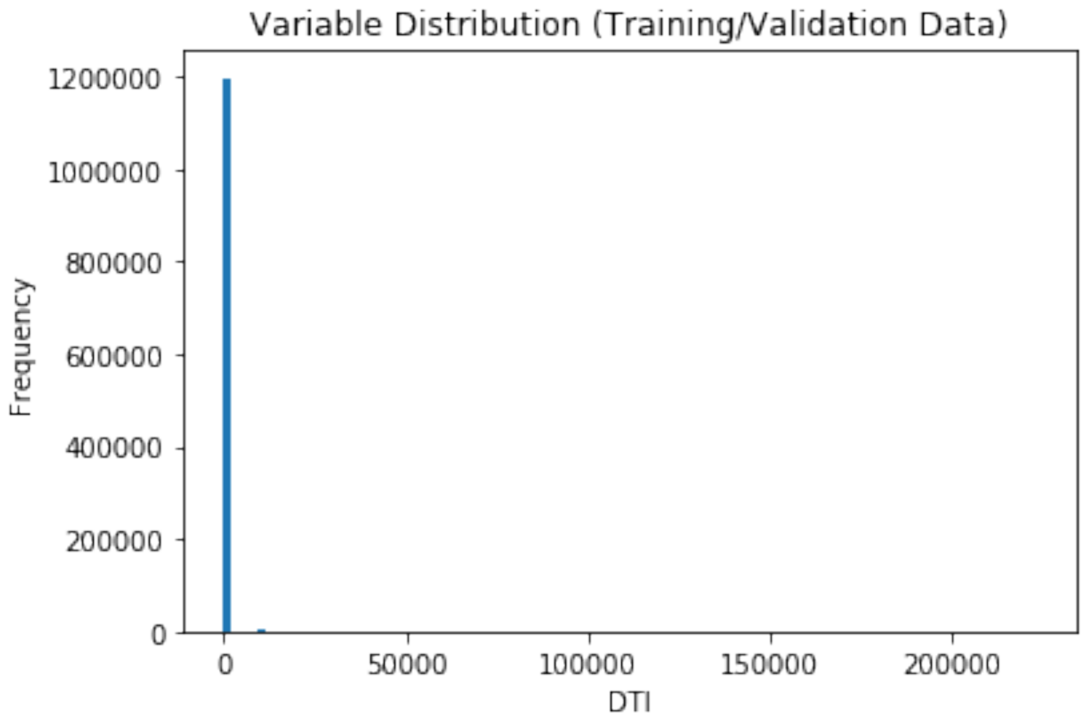
1.4 Label, combine and add dummies

```
In [113]: # Add binary outcome labels
          rej_st["outcome"] = 0
          app_st["outcome"] = 1
          # Combine
          rej_st = rej_st.sample(len(app_st))
          data = pd.concat([rej_st, app_st])

In [114]: data = pd.get_dummies(data, columns=['emp_length', 'state', 'title'], drop_first=True)

In [115]: column_names = data.columns.values
          state_dummies = [_ for _ in column_names if _[:6]=="state_"]
          title_dummies = [_ for _ in column_names if _[:6]=="title_"]
          emp_length_dummies = [_ for _ in column_names if (_[:11]=="emp_length_") and (_ != "em

In [116]: # Shuffle
          data = data.sample(frac=1).reset_index(drop=True)
          #data.to_csv("cleaned_data.csv")
          #df = pd.read_csv("cleaned_data.csv", index_col=0)

          # Split into test and train/validation data
          ratio = 0.8
          cutoff = int(ratio*len(data))
          train_val, test = data[:cutoff], data[cutoff:]
```

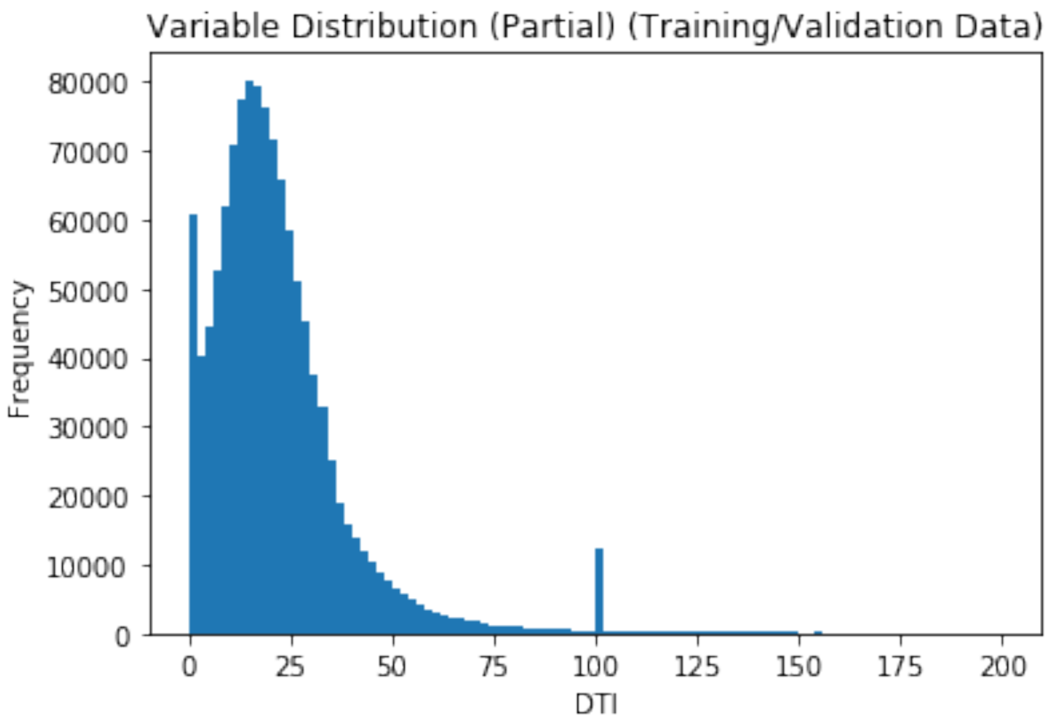
1.5 Examine distributions

1.5.1 DTI ratio

```
In [117]: import matplotlib.pyplot as plt
          temp = train_val["dti"]
          plt.hist(temp,bins=100)
          plt.title("Variable Distribution (Training/Validation Data)")
          plt.xlabel("DTI")
          plt.ylabel("Frequency")
          plt.savefig("DTI.png",dpi=200)
          plt.show()
```



```
In [118]: temp = train_val["dti"][train_val["dti"]<200]
          plt.hist(temp,bins=100)
          plt.title("Variable Distribution (Partial) (Training/Validation Data)")
          plt.xlabel("DTI")
          plt.ylabel("Frequency")
          plt.savefig("DTIpartial.png",dpi=200)
          plt.show()
```



From the graphs above we see that DTI has an extremely long-tailed distribution. When we zoom in to range between 0 and 200, we see that it looks like a partial bell-shaped curve except for high peaks at 0 and 100. I suspect that the peak at 100 is because some people enter 100% when the actual DTI exceeds 100% due to some sort of logistic/psychological reasons.

```
In [119]: train_val = train_val[train_val["dti"] <=100]
```

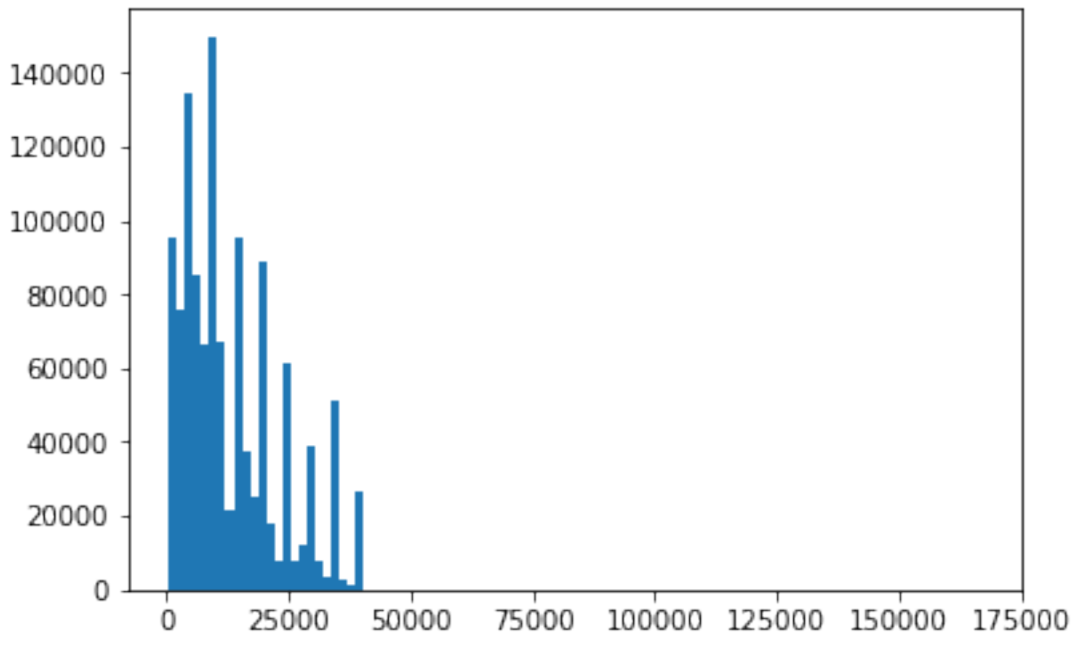
```
In [120]: # Normalize into a range between 0 and 1
          train_val["dti"] = train_val["dti"]/100.
```

```
In [121]: train_val["dti"].describe().round(3)
```

```
Out[121]: count    1181308.000
          mean           0.215
          std            0.163
          min            0.000
          25%            0.110
          50%            0.186
          75%            0.278
          max            1.000
          Name: dti, dtype: float64
```

1.5.2 Amount requsted

```
In [122]: plt.hist(train_val["amount"],bins=100)
          plt.show()
```



```
In [123]: # Adding a log term for later exploratory purpose
          train_val["log_amount"] = train_val["amount"].apply(np.log10)

In [124]: train_val["square_amount"] = train_val["amount"].apply(np.square)
```

1.6 Rescale

```
In [125]: from sklearn.preprocessing import MinMaxScaler, RobustScaler

          amount_scaler = RobustScaler()
          amount_scaler.fit(train_val[["amount"]])
          train_val["amount"] = amount_scaler.transform(train_val[["amount"]])

          log_amount_scaler = MinMaxScaler()
          log_amount_scaler.fit(train_val[["log_amount"]])
          train_val["log_amount"] = log_amount_scaler.transform(train_val[["log_amount"]])

In [126]: square_amount_scaler = RobustScaler()
          square_amount_scaler.fit(train_val[["square_amount"]])
          train_val["square_amount"] = square_amount_scaler.transform(train_val[["square_amount"

In [64]: train_val["amount"].describe().round(3)

Out[64]: count    1181439.000
         mean           0.237
         std            0.668
         min           -0.633
         25%           -0.333
         50%            0.000
         75%            0.667
         max           10.443
         Name: amount, dtype: float64
```

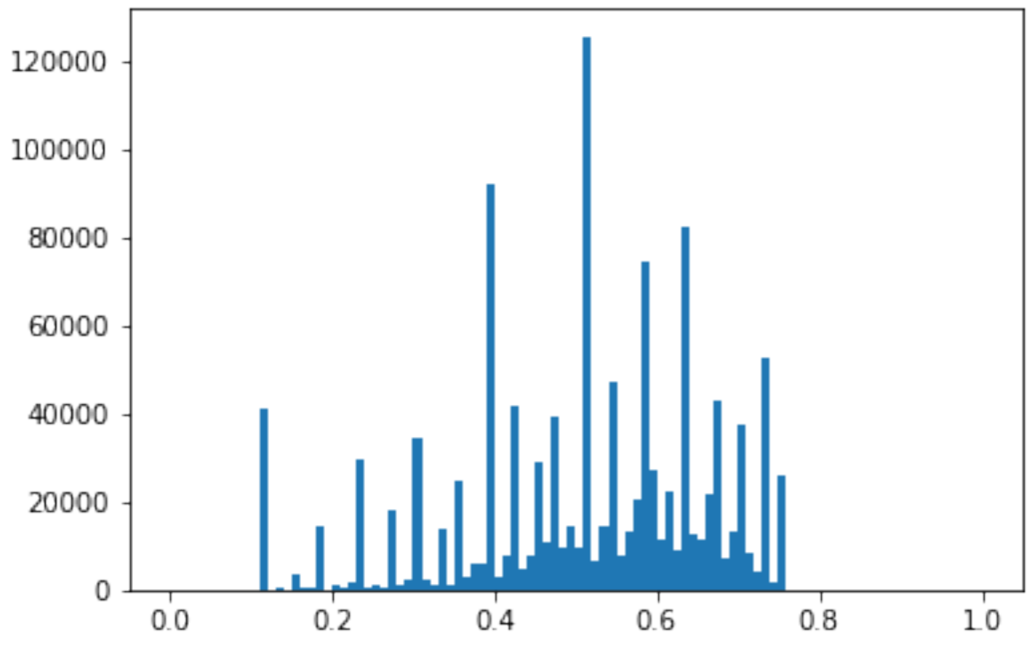
```
In [65]: train_val["log_amount"].describe().round(3)
```

```
Out[65]: count    1181439.000
         mean           0.511
         std            0.155
         min            0.000
         25%            0.396
         50%            0.516
         75%            0.635
         max            1.000
         Name: log_amount, dtype: float64
```

```
In [66]: train_val["square_amount"].describe().round(3)
```

```
Out[66]: count    1181439.000
         mean           0.491
         std            1.011
         min           -0.266
         25%           -0.200
         50%            0.000
         75%            0.800
         max           73.793
         Name: square_amount, dtype: float64
```

```
In [67]: plt.hist(train_val["log_amount"],bins=100)
         plt.show()
```



```
In [127]: # Interaction terms
          int_emp_dummies = []
          for each in emp_length_dummies:
              train_val["int_"+each] = train_val[each]*train_val["amount"]
              int_emp_dummies.append("int_"+each)

          train_val["int_dti"] = train_val["amount"]*train_val["dti"]
```

2 Selecting model

(I tried different combinations of variables but only the code for one version is here because I overrode them each time)

```
In [128]: train_val = train_val.reset_index(drop=True)#prevent error
```

```
In [132]: ## Logistic regression with cross validation
          from sklearn.model_selection import KFold
          from sklearn.linear_model import LogisticRegression
          from sklearn.metrics import mean_squared_error, r2_score

          n_splits = 5
          kf = KFold(n_splits=n_splits)

          #, "log_amount", "dti", "int_dti"]+title_dummies+emp_length_dummies+int_emp_dummies
          X = train_val[["amount","dti", "emp_length_int"]+title_dummies].as_matrix()
          y = train_val["outcome"]

          for train_index, val_index in kf.split(X):
              X_train, X_val = X[train_index], X[val_index]
              y_train, y_val = y[train_index], y[val_index]
              model = LogisticRegression()
              model.fit(X_train, y_train)
              print model.coef_[0][:2]
              predicted = model.predict(X_val)
              print "acc =", np.count_nonzero(y_val == predicted)/float(len(y_val))
```

```
[ 0.20837329 -2.86292671]
acc = 0.78933980073
[ 0.20921845 -2.87609059]
acc = 0.786364290491
[ 0.21469519 -2.87395225]
acc = 0.7883282119
[ 0.2072711  -2.89193638]
acc = 0.789093417873
[ 0.21329209 -2.86350469]
acc = 0.78903839398
```

2.1 Final model

```
In [139]: X = train_val[["amount", "dti", "int_dti"]+title_dummies+emp_length_dummies+int_emp_du
          y = train_val["outcome"]
          model = LogisticRegression()
          model.fit(X, y)
          print model.coef_[0]
          predicted = model.predict(X)
          print "train acc =", np.count_nonzero(y == predicted)/float(len(y))
```

```
[ 0.74446368 -2.0517529  -1.28003768  1.87354053  1.54886887  5.79813354
  1.22503125  0.11645937  1.00011845  0.66478078  0.79614233  0.50539098
  0.04529005  0.44640982  0.71468339  6.27798018  1.14108141  0.57601499
  0.6557604   0.70733303 -3.04605011  1.02536479  0.91460529  0.93721167
  1.24578532 -4.27233204 -0.86562342 -0.51032138 -0.5935448  -0.61977619
 -0.12287561 -0.78824548 -0.88415396 -0.76454078 -0.82781578 -0.21432553]
train acc = 0.896708563728
```

3 Evaluation

```
In [140]: # Perform the same preprocessing as with train_val
          normdti_test = test[test["dti"]<=1.]
          normdti_test = normdti_test.reset_index(drop=True)
```

```
        normdti_test["dti"] = normdti_test["dti"]/100.
        normdti_test["amount"] = amount_scaler.transform(normdti_test[["amount"]])

        # Interaction terms
        for each in emp_length_dummies:
            normdti_test["int_"+each] = normdti_test[each]*normdti_test["amount"]

        normdti_test["int_dti"] = normdti_test["amount"]*normdti_test["dti"]
```

3.1 Accuracy

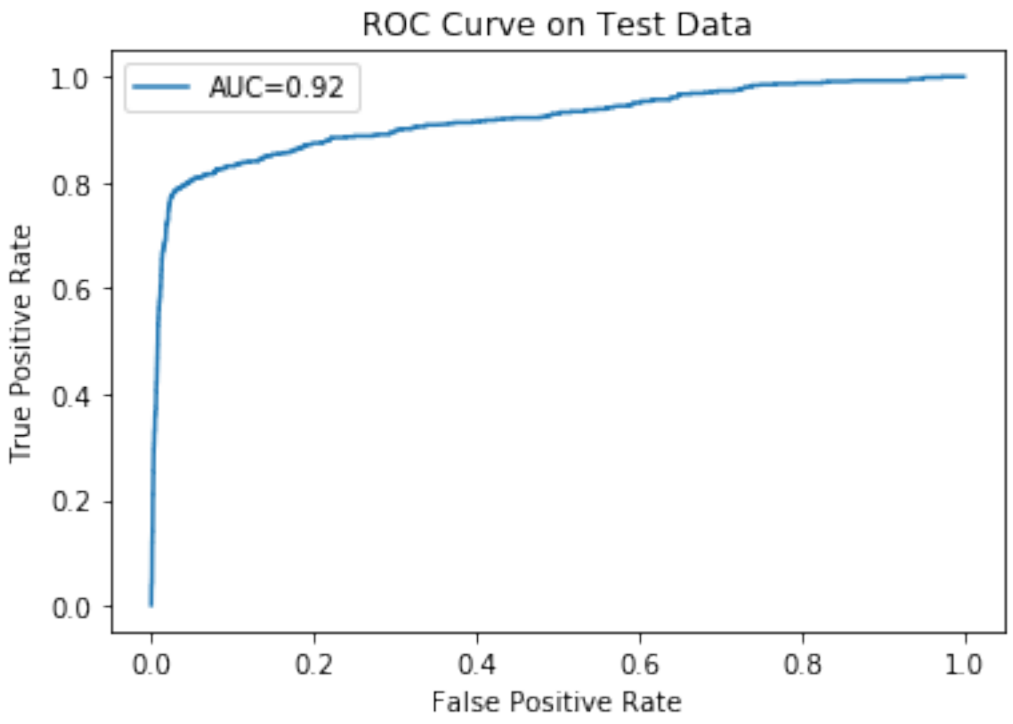
```
In [141]: X_test = normdti_test[["amount", "dti", "int_dti"]+title_dummies+emp_length_dummies+in
          y_test = normdti_test["outcome"]
          predicted = model.predict(X_test)
          print "test acc =", (y_test == predicted).sum()/float(len(y_test))
```

```
test acc = 0.94580402259
```

3.2 ROC Curve

```
In [142]: _, y_prob = zip(*model.predict_proba(X_test))
```

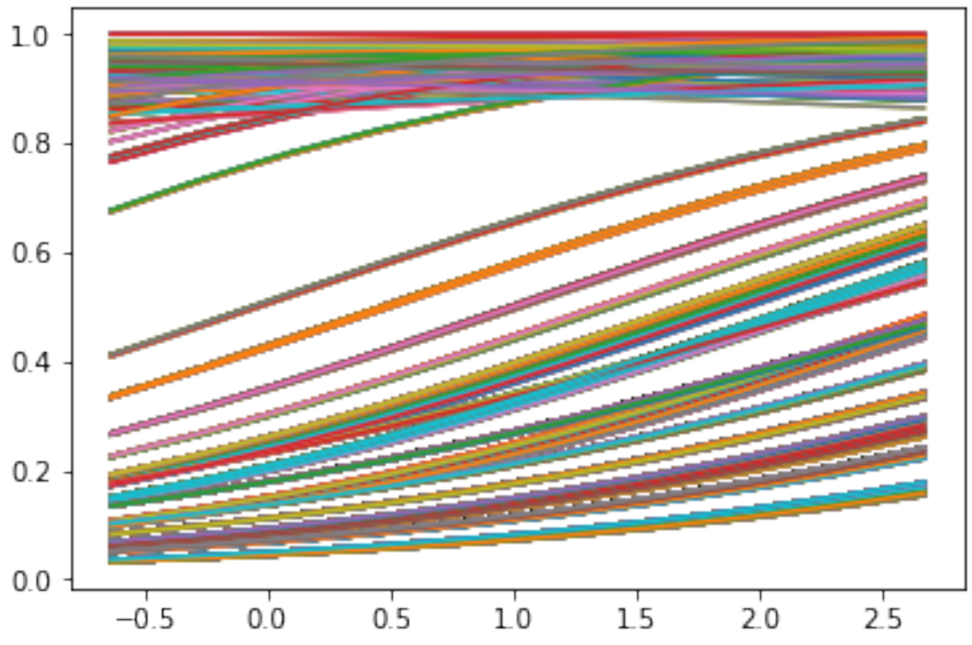
```
In [143]: from sklearn.metrics import roc_curve
          fpr, tpr, thres = roc_curve(y_test, y_prob)
          plt.plot(fpr, tpr, label="AUC={}".format(round(roc_auc_score(y_test, y_prob),2)))
          plt.title("ROC Curve on Test Data")
          plt.xlabel("False Positive Rate")
          plt.ylabel("True Positive Rate")
          plt.legend()
          plt.savefig("ROC.png", dpi=200)
          plt.show()
```



```
In [144]: from sklearn.metrics import roc_auc_score
          print "Area under Curve", roc_auc_score(y_test, y_prob)

Area under Curve 0.918430204598


In [145]: def is_ascending(lis):
              return all(l<=r for l, r in zip(lis[:-1], lis[1:]))

In [148]: amount_list = np.linspace(-0.633, 2.666,10)
          count = 0
          for i in range(10000):
              r = np.random.randint(len(X_test))
              sample = X_test[r]
              outcome = y_test[r]
              pre_lis = []
              for each in amount_list:
                  altered = sample.copy()
                  altered[0]=each
                  #interaction
                  altered[2]=each*altered[1]
                  altered[-len(int_emp_dummies):] = each*sample[-len(int_emp_dummies)*2:-len(int
                  predicted = model.predict_proba([altered])
                  pre_lis.append(round(predicted[0][1],4))
```

```
    plt.plot(amount_list, pre_lis)
    #print pre_lis
    if not is_ascending(pre_lis):
        count += 1
        #print "NOT ASCENDING"
        #print "#",r, "amount=", sample[0], "outcome=", outcome
        #print pre_lis[0],"->", pre_lis[-1]
    else:
        pass
        #print "#",r, "amount=", sample[0], "outcome=", outcome
        #print pre_lis[0],"->", pre_lis[-1]
"""plt.xticks(amount_list[::2], np.int_(amount_scaler.inverse_transform([amount_list])
plt.title("Prediction for 100 Random Individuals")
plt.xlabel("Amount Requested")
plt.ylabel("Predicted Probability of Approval")
plt.savefig("trend.png", dpi=200)
plt.show()"""
print count
```

```
332
```


1.0
0.8
0.6
0.4
0.2
0.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5