CS156 Assignment 2

Yoav Rabinovich, July 2018

We will crease a random forest classifier to distinguish between rejected and accepted loan requests based on features of the requests and the requesting individuals.

In [2]:

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import datetime as dt
import re
```

First we must make the two datasets compatible. We discard features that don't apply to both rejected and accepted loan requests. We discard state information since the information is inherent in the zipcode data. Leaving the state information in would help us estimate predictions for datapoints in rare zipcodes, but will also tend to hide differences between zipcodes inside a given state, which we predict to be salient and useful. We represent the zipcodes as dummy variables to make sure we don't assume a linear relationship between zip code numbers. We discard the loan title, since no unified titling standard was used to help us easily compare different categories, and most titles are unique. We can also discard the policy code, since it's invariant across the rejected sample. We found no analogous field for risk/credit scores in the approved dataset, so we discard risk score. Hopefully debt-to-income ration would covariate and capture that information by proxy. While our dataset is limited and not fully similar in the range of dates, we can better extrapolate to dates not covered by the dataset by considering the seasonal trends divorced from specific years. Since one of our datasets does not provide exact dates, we only preserve the month the request was made. We also want to use the relationship between employment lengths for different applicants, without information on lengths shorter than a year or longer than 9. We approximate less than a year to a zero, and while the average employment length for applicants with 10 or more years of experience is probably much higher than 10, guessing a value would be risky and they are rare enough that we can safely set their employment length at 10 years. This leaves us with the features:

- Amount Requested
- Application Date (month*)
- Debt-To-Income Ratio
- Zip Code (as dummy variables)
- Employment Length (0-10)

*To best represent the cyclical nature of the yearly cycle, we convert the month values to two variables capturing sin and cos. This way Dec and Jan are as similar as Jan and Feb.

In [3]:

```
# Importing the datasets, erased first and last few unformatted rows manually, as well as 3
#taking a subset of 3000 rejected examples to ease computation
approved = pd.read_csv('LoanStats3A.csv')
rejected = pd.read_csv('RejectStatsA.csv')
rejected = rejected.iloc[:3000,:]
#extracting relevant features
rejected = rejected.iloc[:,[0,1,4,5,7]]
approved = pd.concat([approved.iloc[:,2],approved.iloc[:,15],approved.iloc[:,24],approved.i
rejected.columns=["loan_amnt","issue_d","dti","zip_code","emp_length"]
```

```
D:\Program Files\Anaconda2\lib\site-packages\IPython\core\interactiveshell.p
y:2717: DtypeWarning: Columns (47) have mixed types. Specify dtype option on
import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

In [4]:

```
#converting dates to month variables
months = pd.to_datetime(rejected['issue_d']).dt.strftime('%m').astype(float)
msin=np.sin(2.*np.pi*months/12.).astype(float)
mcos=np.cos(2.*np.pi*months/12.).astype(float)
rejected.drop(['issue_d'],axis=1,inplace=True)
rejected=pd.concat([rejected,pd.DataFrame({"msin":msin,"mcos":mcos})],axis=1)
#converting dti
rejected['dti'] = rejected['dti'].str.rstrip('%')
rejected.head()
```

Out[4]:

	loan_amnt	dti	zip_code	emp_length	mcos	msin
0	1000.0	10	481xx	4 years	-0.866025	0.5
1	1000.0	10	010xx	< 1 year	-0.866025	0.5
2	11000.0	10	212xx	1 year	-0.866025	0.5
3	6000.0	38.64	017xx	< 1 year	-0.866025	0.5
4	1500.0	9.43	209xx	< 1 year	-0.866025	0.5

In [5]:

```
#dates for accepted
months = approved['issue_d']
for index,date in months.iteritems():
    if (str.startswith(date,"7-"))or(str.startswith(date,"8-"))or(str.startswith(date,"9-")
        months.set_value(index,"0"+date)
months = pd.to_datetime(months,format="%y-%b").dt.strftime("%m").astype(float)
msin=np.sin(2.*np.pi*months/12.).astype(float)
mcos=np.cos(2.*np.pi*months/12.).astype(float)
approved.drop(['issue_d'],axis=1,inplace=True)
approved=pd.concat([approved,pd.DataFrame({"msin":msin,"mcos":mcos})],axis=1)
approved.head()
```

Out[5]:

	loan_amnt	dti	zip_code	emp_length	mcos	msin
0	5000	27.65	860xx	10+ years	1.0	-2.449294e-16
1	2500	1.00	309xx	< 1 year	1.0	-2.449294e-16
2	2400	8.72	606xx	10+ years	1.0	-2.449294e-16
3	10000	20.00	917xx	10+ years	1.0	-2.449294e-16
4	3000	17.94	972xx	1 year	1.0	-2.449294e-16

In [6]:

```
#we can now combine the datasets and continue to process together, labeling approval as 1 a
approved.insert(0, 'label', 1)
rejected.insert(0, 'label', 0)
data = pd.concat([approved,rejected])
```

Employment length has many n/a values. We could use mean imputation to approximate them, making the slightly precarious asssumption that missing values don't correlate with other observables in the data. We could also use expectation maximization to cluster them into 11 values (0-10) by approximation using similarity with labeled data. However, since an option for unemployed applicants is missing, it's likely that this is what ost n/a values in the data represent, so we'll be setting them to -1 to fit the relationship we've established.

In [7]:

```
#setting length to 0-10
def emplength(x):
    if "<" in x: return 0
    elif "/" in x: return -1
    else: return re.findall("\d+", x)[0]
data['emp_length'] = data['emp_length'].astype(str).apply(emplength)
```

In [8]:

```
#setting dummy variables for zip code
data['zip_code']=data["zip_code"].str.rstrip('xx')
data = pd.concat([data,pd.get_dummies(data['zip_code'])],axis=1)
data.drop(['zip_code'],axis=1,inplace=True)
#attach yet-to-be-encountered dummies
rang = ["%03d" % i for i in range(1000)]
for ele in rang:
    if ele not in data.columns:
        spot=int(ele)+6
        data.insert(spot,ele,0.)
data.head()
```

Out[8]:

	label	loan_amnt	dti	emp_length	mcos	msin	000	001	002	003	...	990	9
0	1	5000.0	27.65	10	1.0	-2.449294e-16	0.0	0.0	0.0	0.0	...	0.0	0
1	1	2500.0	1	0	1.0	-2.449294e-16	0.0	0.0	0.0	0.0	...	0.0	0
2	1	2400.0	8.72	10	1.0	-2.449294e-16	0.0	0.0	0.0	0.0	...	0.0	0
3	1	10000.0	20	10	1.0	-2.449294e-16	0.0	0.0	0.0	0.0	...	0.0	0
4	1	3000.0	17.94	1	1.0	-2.449294e-16	0.0	0.0	0.0	0.0	...	0.0	0

5 rows × 1006 columns

Now that the data's processed, we can proceed with classification. I chose to try a new method to expand my horizons, and settled on the famed random forest bagging approach. It combines the decisions of multiple decision trees to reach an agreed conclusion, and preforms classification tasks well. It's important to scale our features before using Random Forests, to prevent the differnece in variance scales to let some features dominate over others.

In [9]:

```
#Classification

#Splitting the dataset into the Training set and Test set
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data.iloc[:,1:], data.iloc[:,0], test_s


# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print(X_train[:,0:10])
```

```
D:\Program Files\Anaconda2\lib\site-packages\sklearn\cross_validation.py:41:
DeprecationWarning: This module was deprecated in version 0.18 in favor of t
he model_selection module into which all the refactored classes and function
s are moved. Also note that the interface of the new CV iterators are differ
ent from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)

[[-0.11744222 -0.01575715 -0.72579424 ...  0.          0.
   0.        ]
 [-0.38582524 -0.00884043  1.46192619 ...  0.          0.
   0.        ]
 [ 0.32538975 -0.01538826  0.64153103 ...  0.          0.
   0.        ]
 ...
 [-0.78839976 -0.01653593 -0.99925929 ...  0.          0.
   0.        ]
 [ 0.4193238  -0.00964994  1.46192619 ...  0.          0.
   0.        ]
 [ 0.15094079 -0.00627868  0.09460092 ...  0.          0.
   0.        ]]
```

In [10]:

```
# Fitting Random Forest Classification to the Training set
from sklearn.ensemble import RandomForestClassifier

#CV
from sklearn.model_selection import GridSearchCV
parameters_n = {'n_estimators': [2, 5, 7]}
classifier = GridSearchCV(RandomForestClassifier(), parameters_n,cv=3)

#fitting
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

#Classification Report
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred, target_names=["Approved","Rejected"]))
```

```
             precision    recall  f1-score   support

   Approved       0.75      0.38      0.50       800
   Rejected       0.95      0.99      0.97     10584

avg / total       0.94      0.95      0.94     11384


AttributeErrorTraceback (most recent call last)
<ipython-input-10-26ac7f629c1d> in <module>()
     17 print(classification_report(y_test, y_pred, target_names=["Approved"
,"Rejected"]))
     18
---> 19 importances = classifier.feature_importances_
     20 std = np.std([classifier.feature_importances_ for tree in classifie
r.estimators_],
     21              axis=0)

AttributeError: 'GridSearchCV' object has no attribute 'feature_importances
_'
```

In [13]:

```
#Feature importances
importances = classifier.best_estimator_.feature_importances_
std = np.std([classifier.best_estimator_.feature_importances_ for tree in classifier.best_e
             axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking (best 10):")

for f in range(10):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))
```

```
Feature ranking (best 10):
1. feature 1 (0.315340)
2. feature 0 (0.144529)
3. feature 2 (0.081250)
4. feature 4 (0.055665)
5. feature 3 (0.040443)
6. feature 686 (0.003499)
7. feature 305 (0.003331)
8. feature 117 (0.003138)
9. feature 307 (0.002967)
10. feature 335 (0.002839)
```

Interestingly we see that the DTI ratio is more important than the loan amount. Now that we have an accurate model, we can use the predict function to find the highest loan amount a specific applicant can expect to get approved. We set the applicant's qualities arbitratily here, and iterate over the loan amount feature to find the classification boundary.

In [32]:

```
#takes features except loan amount and a classifier and returns boundary for approval
def approval_boundary(dti,zip_code,emp_length,month,clf,step=100,max_steps=1000):
    #creates datapoint from features
    x = np.zeros(1005)
    x[1]=dti
    x[2]=emp_length
    x[3]=np.cos(2.*np.pi*month/12.).astype(float)
    x[4]=np.sin(2.*np.pi*month/12.).astype(float)
    x[zip_code+5]=1
    #initalize loan amount
    x[0]=50000
    #feature scaling
    [y] = sc.transform([x])
    print[y]
    if clf.predict([y])==0:
        print("No chance!")
        return
    #loop with increasing amount
    for i in range(max_steps):
        if clf.predict([y])==1:
            x[0]+=step
            [y] = sc.transform([x])
            continue
        else:
            print("Approval boundary: $" + str(x[0]) + " +/- $" + str(step))
            return
```

In [33]:

```
#Example
approval_boundary(80,300,1,12,classifier)
```

```
[array([ 5.25021802,  0.06187409, -0.99925929, ..., -0.02231675,
       -0.01530714, -0.00541134])]
```

In [ ]:

```

```