Bank_failure_rand_forests_m2_ex2

October 25, 2018

0.1 Modeling of bank failures by FDIC

In this assignment you will be using: - Decision Trees - Random Forests - Boosted Trees

All in the context of classification problem as applied to bank defaults data set. Let's get started!

0.2 About iPython Notebooks

iPython Notebooks are interactive coding environments embedded in a webpage. You will be using iPython notebooks in this class. You only need to write code between the ### START CODE HERE ### and ### END CODE HERE ### comments. After writing your code, you can run the cell by either pressing "SHIFT"+"ENTER" or by clicking on "Run Cell" (denoted by a play symbol) in the upper bar of the notebook.

We will often specify "(X lines of code)" in the comments to tell you about how much code you need to write. It is just a rough estimate, so don't feel bad if your code is longer or shorter.

In [17]:
```
import pandas as pd
import numpy as np
import time

import os
import functools
import math
import random
import sys, getopt

sys.path.append("..")
import grading

try:
    import matplotlib.pyplot as plt
    %matplotlib inline
except:
    pass
```

In [18]:
```
### ONLY FOR GRADING. DO NOT EDIT ###
submissions=dict()
assignment_key="HnqO7_GcEeeQcBJXIhP2bA"
```

```
        all_parts=["Pb9kd", "ZdjyW", "IfVpy","Tifr3","X8djk"]
        ### ONLY FOR GRADING. DO NOT EDIT ###
```

In [51]:
```
        # COURSERA_TOKEN = # the key provided to the Student under his/her email on submission
        # COURSERA_EMAIL = # the email
        COURSERA_TOKEN="WjrjMOpi7rJMF1uU"
        COURSERA_EMAIL="cilsya@yahoo.com"
```

In [20]:
```
        # common cell - share this across notebooks
        state_cols = ['log_TA','NI_to_TA', 'Equity_to_TA', 'NPL_to_TL', 'REO_to_TA',
                      'ALLL_to_TL', 'core_deposits_to_TA', 'brokered_deposits_to_TA',
                      'liquid_assets_to_TA', 'loss_provision_to_TL',
                      'ROA',
                      'NIM', 'assets_growth']

        # Macro Economic Variables (MEVs)
        all_MEVs = np.array(['term_spread',
                             'stock_mkt_growth',
                             'real_gdp_growth',
                             'unemployment_rate_change',
                             'treasury_yield_3m',
                             'bbb_spread',
                             'bbb_spread_change'])

        MEV_cols = all_MEVs.tolist()

        next_state_cols = ['log_TA_plus_1Q','NI_to_TA_plus_1Q', 'Equity_to_TA_plus_1Q', 'NPL_to
                           'ALLL_to_TL_plus_1Q', 'core_deposits_to_TA_plus_1Q', 'brokered_depos
                           'liquid_assets_to_TA_plus_1Q', 'loss_provision_to_TL_plus_1Q',
                           'ROA_plus_1Q',
                           'NIM_plus_1Q',
                           'assets_growth_plus_1Q',
                           'FDIC_assessment_base_plus_1Q_n']
```

In [21]:
```
        df_train = pd.read_hdf(os.getcwd() + '/data/df_train_FDIC_defaults_1Y.h5', key='df')
        df_test = pd.read_hdf(os.getcwd() + '/data/df_test_FDIC_defaults_1Y.h5', key='df')
        df_data = pd.read_hdf(os.getcwd() + '/data/data_adj_FDIC_small.h5', key='df')
        df_closure_learn = pd.read_hdf(os.getcwd() + '/data/df_FDIC_learn.h5', key='df')

        df_all_defaulters_in_1Y = df_closure_learn[df_closure_learn.defaulter == 1].reset_index
        selected_dates = df_all_defaulters_in_1Y.date.unique()
        defaulted_banks = df_all_defaulters_in_1Y['IDRSSD'].unique()
        print('Number of unique dates on which defaulted within 1-st year %d'  % len(selected_d
        print('Number of unique banks defaulted within 1-st year %d' % len(defaulted_banks))

        # failure dates
        # df_data[df_data['Failure / Assistance '].notnull()].date.value_counts()
```

```
Number of unique dates on which defaulted within 1-st year 39
```

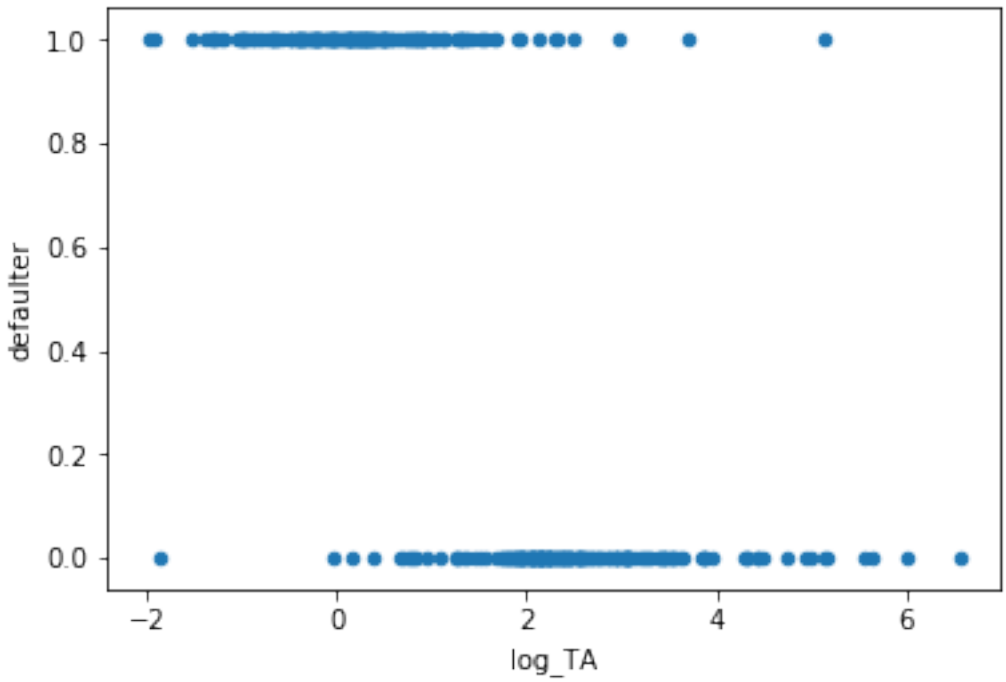
```
Number of unique banks defaulted within 1-st year 472
```

```
In [22]: # function for a flexible way to make the train and test sets
         def make_train_and_test(df_in, perc_train=66.0, split_by_IDRSSD=True):

             reset_index_flag=False
             df = df_in.copy()
             if 'IDRSSD' in df.index.names:
                 reset_index_flag = True
                 df.reset_index(inplace=True)

             len_df = len(df)
             len_df_train = int(np.floor(0.01*perc_train*len_df))

             if split_by_IDRSSD == True:
                 # split by names
                 unique_IDRSSD = df.IDRSSD.unique()
                 num_unique_IDRSSD = len(unique_IDRSSD)
                 num_IDRSSD_train = int(np.floor(0.01*perc_train*num_unique_IDRSSD))

                 # re-shuffle the list of IDRSSD
                 np.random.shuffle(unique_IDRSSD)
                 IDRSSD_train = unique_IDRSSD[0:num_IDRSSD_train]
                 IDRSSD_test = unique_IDRSSD[num_IDRSSD_train:]

                 df_train = df[np.in1d(df.IDRSSD, IDRSSD_train)].copy()
                 df_test = df[np.in1d(df.IDRSSD, IDRSSD_test)].copy()

             elif split_by_IDRSSD == False:
                 # split by rows

                 idx = np.arange(len_df)
                 np.random.shuffle(idx)
                 df_train = df.ix[idx[0:len_df_train]]
                 df_test = df.ix[idx[len_df_train:]]

             return df_train, df_test
```

0.2.1 Visualize binary classification data

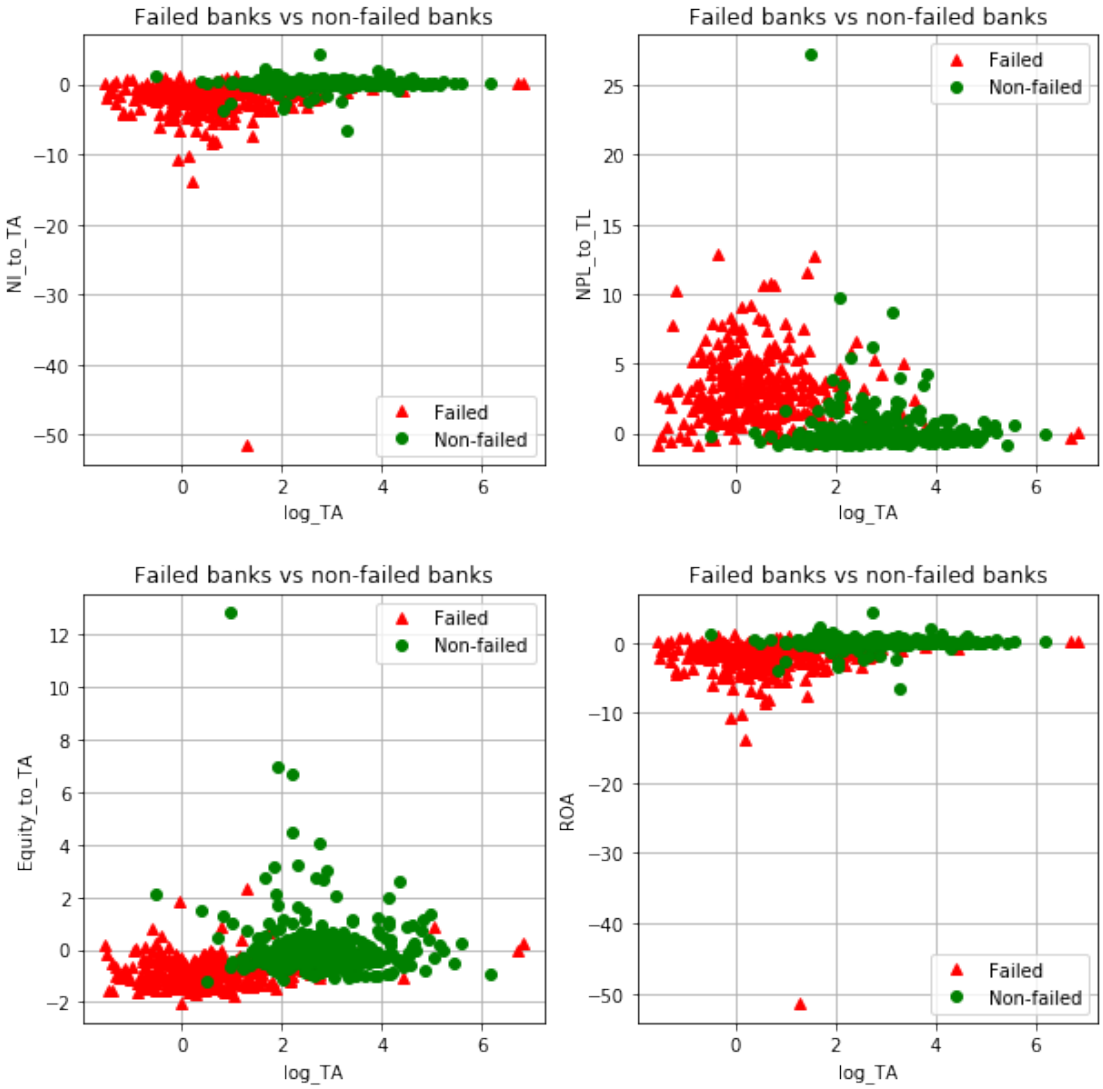
```
In [23]: df_test.plot(x=state_cols[0], y='defaulter', kind='scatter')
```

```
Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6cf96bc390>
```



In [24]:
```
# Plot 4 scatter plots together

# log_TA / NI_to_TA
# log_TA / NPL_to_TL
# log_TA / Equity_to_TA
# log_TA / ROA

first_indx = [0, 0, 0, 0]
second_indx = [1, 3, 2, 10]

X_train = df_train[state_cols].values
y_train = df_train.defaulter.values # .reshape(-1,1)

num_plots = 4
if num_plots % 2 == 0:
    f, axs = plt.subplots(num_plots // 2, 2)
else:
    f, axs = plt.subplots(num_plots// 2 + 1, 2)

f.subplots_adjust(hspace=.3)
f.set_figheight(10.0)
f.set_figwidth(10.0)

for i in range(num_plots):
```

```
        if i % 2 == 0:
            first_idx = i // 2
            second_idx = 0
        else:
            first_idx = i // 2
            second_idx = 1

        axs[first_idx,second_idx].plot(X_train[y_train == 1.0, first_indx[i]],
                                       X_train[y_train == 1.0, second_indx[i]], 'r^', label
        axs[first_idx,second_idx].plot(X_train[y_train == 0.0, first_indx[i]],
                                       X_train[y_train == 0.0, second_indx[i]], 'go', label

        axs[first_idx, second_idx].legend()
        axs[first_idx, second_idx].set_xlabel('%s' % state_cols[first_indx[i]])
        axs[first_idx, second_idx].set_ylabel('%s' % state_cols[second_indx[i]])
        axs[first_idx, second_idx].set_title('Failed banks vs non-failed banks')
        axs[first_idx, second_idx].grid(True)

    if num_plots % 2 != 0:
        f.delaxes(axs[i // 2, 1])

    # plt.savefig('Failed_vs_nonfailed_rr_plot.png')
```




```
In [25]: state_cols
```

```
Out[25]: ['log_TA',
          'NI_to_TA',
          'Equity_to_TA',
          'NPL_to_TL',
          'REO_to_TA',
          'ALLL_to_TL',
          'core_deposits_to_TA',
          'brokered_deposits_to_TA',
          'liquid_assets_to_TA',
          'loss_provision_to_TL',
          'ROA',
          'NIM',
          'assets_growth']
```

In [26]:
```
# Column 'ROA' is redundant, it is the same as NI_to_TA, so remove it
state_cols = [c for c in state_cols if c != 'ROA']

print(state_cols)
print('Len state_cols = ', len(state_cols))
```

```
['log_TA', 'NI_to_TA', 'Equity_to_TA', 'NPL_to_TL', 'REO_to_TA', 'ALLL_to_TL', 'core_deposits_to
Len state_cols =  12
```

In [27]:
```
# make the train and test datasets
choice = 0  # selection of predictors. Add tangible equity and assessment base as predi
predict_within_1Y = False # True

if choice == -1: # only state cols
    cols = state_cols
elif choice == 0:  # original variables
    cols = state_cols + MEV_cols

trX = df_train[cols].values
teX = df_test[cols].values
num_features = len(cols)

if predict_within_1Y == True:
    trY = df_train[['default_within_1Y','no_default_within_1Y']].values
    teY = df_test[['default_within_1Y','no_default_within_1Y']].values
else:
    trY = df_train[['defaulter','non_defaulter']].values
    teY = df_test[['defaulter','non_defaulter']].values

num_classes = 2
num_components = len(cols)
```

In [28]:
```
# look at correlations
df_train[MEV_cols].corr()
```

Out[28]:

	term_spread	stock_mkt_growth	real_gdp_growth	\
term_spread	1.000000	0.002993	-0.145941	
stock_mkt_growth	0.002993	1.000000	-0.148941	
real_gdp_growth	-0.145941	-0.148941	1.000000	
unemployment_rate_change	0.299972	0.461947	-0.825802	
treasury_yield_3m	-0.633991	-0.081915	0.041596	
bbb_spread	0.392349	0.417379	-0.820518	
bbb_spread_change	-0.465767	-0.762702	0.385007	

	unemployment_rate_change	treasury_yield_3m	\
term_spread	0.299972	-0.633991	
stock_mkt_growth	0.461947	-0.081915	
real_gdp_growth	-0.825802	0.041596	

```
unemployment_rate_change                  1.000000          0.034355
treasury_yield_3m                         0.034355          1.000000
bbb_spread                                0.881223         -0.272072
bbb_spread_change                        -0.657093          0.290414

                            bbb_spread  bbb_spread_change
term_spread                   0.392349          -0.465767
stock_mkt_growth              0.417379          -0.762702
real_gdp_growth              -0.820518           0.385007
unemployment_rate_change      0.881223          -0.657093
treasury_yield_3m            -0.272072           0.290414
bbb_spread                    1.000000          -0.716249
bbb_spread_change            -0.716249           1.000000
```

In [29]:
```
print(teY[:, 0].sum())
print(df_test.defaulter.sum())
print('num_components: %d' % num_components)
state_cols
```

```
161.0
161.0
num_components: 19
```

Out[29]:
```
['log_TA',
 'NI_to_TA',
 'Equity_to_TA',
 'NPL_to_TL',
 'REO_to_TA',
 'ALLL_to_TL',
 'core_deposits_to_TA',
 'brokered_deposits_to_TA',
 'liquid_assets_to_TA',
 'loss_provision_to_TL',
 'NIM',
 'assets_growth']
```

In [30]:
```
from sklearn import neighbors, linear_model
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier

col_start = 0 # 19
comp_to_keep = num_components
```

In [31]:
```
# Do Logistic Regression with a smaller number of predictor, based on analysis of P-val
# for the logistic regression with a full set of variables

# the original set of predictors
# cols_to_use = state_cols + MEV_cols
```