FRE-7773A Final Project _Jiangguangyu_Xue_jx1021

July 9, 2019

1 Final Project 7771

2 Import Data and a brief look on data

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
%matplotlib inline
from datetime import datetime
from sklearn.linear_model import LogisticRegression
import seaborn as sns
import tensorflow as tf
from tensorflow.python.framework import ops
```

```
C:\Users\olive\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conversion of th
  from ._conv import register_converters as _register_converters
```

In [2]:
```
data=pd.read_csv("./final_project_data.csv")
data=data.dropna()
data.index=range(len(data))
```

In [5]:
```
data.shape
```

Out[5]:
```
(3547259, 43)
```

In [4]:
```
data.columns
```

Out[4]:
```
Index(['date', 'id', 'industry', 'ret_raw', 'flag', 'ret_20_raw', 'flag2',
       'ret_raw_norm', 'ret_20_raw_norm', 'ret_raw_norm_lag_21',
       'ret_raw_norm_lag_22', 'ret_raw_norm_lag_23', 'ret_raw_norm_lag_24',
       'ret_raw_norm_lag_25', 'ret_raw_norm_lag_26', 'ret_raw_norm_lag_27',
       'ret_raw_norm_lag_28', 'ret_raw_norm_lag_29', 'ret_raw_norm_lag_30',
       'ret_raw_norm_lag_31', 'ret_raw_norm_lag_32', 'ret_raw_norm_lag_33',
       'ret_raw_norm_lag_34', 'ret_raw_norm_lag_35', 'ret_raw_norm_lag_36',
       'ret_raw_norm_lag_37', 'ret_raw_norm_lag_38', 'ret_raw_norm_lag_39',
       'ret_raw_norm_lag_40', 'ret_20_raw_norm_lag_41_60',
```

```
       'ret_20_raw_norm_lag_61_80', 'ret_20_raw_norm_lag_81_100',
       'ret_20_raw_norm_lag_101_120', 'ret_20_raw_norm_lag_121_140',
       'ret_20_raw_norm_lag_141_160', 'ret_20_raw_norm_lag_161_180',
       'ret_20_raw_norm_lag_181_200', 'ret_20_raw_norm_lag_201_220',
       'ret_20_raw_norm_lag_221_240', 'ret_20_raw_norm_lag_241_260',
       'ret_20_raw_norm_lag_261_280', 'isJan', 'target'],
      dtype='object')
```

In [5]:
```
data.head()
```

Out[5]:
```
       date             id  industry   ret_raw  flag  ret_20_raw  flag2  \
0  20080214    A US Equity      3520 -0.034888  True   -0.115785  False
1  20080214   AA US Equity      1510 -0.004224  True    0.160461  False
2  20080214  AAN US Equity      2550  0.000000  True    0.102639  False
3  20080214 AAON US Equity      2010 -0.046684  True    0.013005  False
4  20080214  AAP US Equity      2550 -0.040387  True    0.064812  False

   ret_raw_norm  ret_20_raw_norm  ret_raw_norm_lag_21   ...    \
0     -0.626602        -1.155291            -0.342618   ...
1      0.538586         1.116039            -0.616964   ...
2      0.699089         0.640614             0.803338   ...
3     -1.074841        -0.096364            -0.091706   ...
4     -0.835580         0.329595             0.635282   ...

   ret_20_raw_norm_lag_121_140  ret_20_raw_norm_lag_141_160  \
0                    -0.927360                     0.184879
1                    -0.811019                     0.657379
2                     1.306399                    -2.065115
3                     0.782537                    -0.145668
4                    -0.844002                    -0.495810

   ret_20_raw_norm_lag_161_180  ret_20_raw_norm_lag_181_200  \
0                     0.157539                     1.093994
1                    -0.466201                     1.449841
2                     0.532074                    -0.593705
3                     1.478548                     1.629201
4                    -0.335421                    -0.517209

   ret_20_raw_norm_lag_201_220  ret_20_raw_norm_lag_221_240  \
0                    -0.189996                     0.915039
1                     0.170922                     0.042121
2                     0.523305                    -0.527815
3                    -1.186070                    -1.054264
4                     0.466003                     0.442235

   ret_20_raw_norm_lag_241_260  ret_20_raw_norm_lag_261_280  isJan  target
0                    -0.259174                    -1.240792    0.0     0.0
1                     0.339868                     1.037797    0.0     1.0
```

```
2                     -1.058531                      0.101127    0.0     1.0
3                      0.029782                      0.089869    0.0     0.0
4                     -0.078167                      0.179575    0.0     1.0

[5 rows x 43 columns]
```

```
In [169]: data[0:10000].groupby("target").count()
```

```
Out[169]:         date    id  industry  ret_raw  flag  ret_20_raw  flag2  ret_raw_norm  \
          target
          0.0     4953  4953      4953     4953  4953        4953   4953          4953
          1.0     5047  5047      5047     5047  5047        5047   5047          5047

                  ret_20_raw_norm  ret_raw_norm_lag_21  ...    \
          target                                        ...
          0.0                4953                 4953  ...
          1.0                5047                 5047  ...

                  ret_20_raw_norm_lag_101_120  ret_20_raw_norm_lag_121_140  \
          target
          0.0                            4953                         4953
          1.0                            5047                         5047

                  ret_20_raw_norm_lag_141_160  ret_20_raw_norm_lag_161_180  \
          target
          0.0                            4953                         4953
          1.0                            5047                         5047

                  ret_20_raw_norm_lag_181_200  ret_20_raw_norm_lag_201_220  \
          target
          0.0                            4953                         4953
          1.0                            5047                         5047

                  ret_20_raw_norm_lag_221_240  ret_20_raw_norm_lag_241_260  \
          target
          0.0                            4953                         4953
          1.0                            5047                         5047

                  ret_20_raw_norm_lag_261_280  isJan
          target
          0.0                            4953   4953
          1.0                            5047   5047

          [2 rows x 42 columns]
```

```
In [6]: data.loc[0:10000,'ret_raw_norm_lag_21':'isJan'].describe()
```

```
Out[6]:        ret_raw_norm_lag_21  ret_raw_norm_lag_22  ret_raw_norm_lag_23  \
        count         10001.000000         10001.000000         10001.000000
```

```
mean          0.002945           0.007331           0.002609
std           0.987706           0.996535           0.980898
min         -13.602740         -13.602740         -14.861496
25%          -0.533603          -0.527397          -0.542848
50%          -0.042586          -0.019447          -0.031049
75%           0.491813           0.516340           0.520341
max          12.711196          12.711196          12.711196

       ret_raw_norm_lag_24  ret_raw_norm_lag_25  ret_raw_norm_lag_26  \
count         10001.000000         10001.000000         10001.000000
mean              0.002278            -0.000843             0.000437
std               0.980745             0.979509             0.975704
min             -14.861496           -14.861496           -14.861496
25%              -0.537156            -0.510389            -0.492754
50%              -0.016208            -0.011861             0.005390
75%               0.533492             0.523377             0.528234
max              12.711196            12.711196            11.536464

       ret_raw_norm_lag_27  ret_raw_norm_lag_28  ret_raw_norm_lag_29  \
count         10001.000000         10001.000000         10001.000000
mean             -0.004543            -0.003967            -0.001801
std               0.975510             0.964357             0.963921
min             -14.861496           -14.861496           -14.861496
25%              -0.492713            -0.490445            -0.496837
50%               0.006594             0.005617             0.010000
75%               0.528967             0.535571             0.545642
max              11.536464             7.349263             7.349263

       ret_raw_norm_lag_30  ...  ret_20_raw_norm_lag_101_120  \
count         10001.000000  ...                 10001.000000
mean             -0.003810  ...                     0.022295
std               0.964916  ...                     0.975781
min             -14.861496  ...                    -3.649990
25%              -0.501826  ...                    -0.443190
50%               0.005694  ...                    -0.019599
75%               0.542263  ...                     0.448565
max               7.521026  ...                    21.233928

       ret_20_raw_norm_lag_121_140  ret_20_raw_norm_lag_141_160  \
count                 10001.000000                 10001.000000
mean                      0.028554                     0.023515
std                       0.954068                     0.959789
min                      -4.702042                    -4.551659
25%                      -0.508246                    -0.532119
50%                       0.001069                    -0.006308
75%                       0.528561                     0.551520
max                       7.543090                     6.937105
```

```
       ret_20_raw_norm_lag_161_180  ret_20_raw_norm_lag_181_200  \
count                 10001.000000                 10001.000000
mean                     -0.003810                     0.007704
std                       0.964149                     0.978488
min                      -4.963919                    -6.385819
25%                      -0.574381                    -0.572143
50%                      -0.084170                    -0.083193
75%                       0.486248                     0.451866
max                       5.565284                     5.969617

       ret_20_raw_norm_lag_201_220  ret_20_raw_norm_lag_221_240  \
count                 10001.000000                 10001.000000
mean                      0.023496                     0.018387
std                       0.971273                     0.953070
min                      -5.619903                    -5.093598
25%                      -0.557106                    -0.527288
50%                      -0.056765                    -0.043638
75%                       0.519637                     0.497415
max                       6.849358                     5.977391

       ret_20_raw_norm_lag_241_260  ret_20_raw_norm_lag_261_280    isJan
count                 10001.000000                 10001.000000  10001.0
mean                      0.011454                    -0.002137      0.0
std                       0.975026                     0.963197      0.0
min                      -4.657949                    -5.345205      0.0
25%                      -0.528431                    -0.573352      0.0
50%                      -0.028611                    -0.035096      0.0
75%                       0.481846                     0.512978      0.0
max                       5.721677                     5.664229      0.0

[8 rows x 33 columns]
```

2.0.1 33 Model Input Features (momentum driven features):

'ret_raw_norm_lag_21', 'ret_raw_norm_lag_22', 'ret_raw_norm_lag_23', 'ret_raw_norm_lag_24', 'ret_raw_norm_lag_25', 'ret_raw_norm_lag_26', 'ret_raw_norm_lag_27', 'ret_raw_norm_lag_28', 'ret_raw_norm_lag_29', 'ret_raw_norm_lag_30', 'ret_raw_norm_lag_31', 'ret_raw_norm_lag_32', 'ret_raw_norm_lag_33', 'ret_raw_norm_lag_34', 'ret_raw_norm_lag_35', 'ret_raw_norm_lag_36', 'ret_raw_norm_lag_37', 'ret_raw_norm_lag_38', 'ret_raw_norm_lag_39', 'ret_raw_norm_lag_40', 'ret_20_raw_norm_lag_41_60', 'ret_20_raw_norm_lag_61_80', 'ret_20_raw_norm_lag_81_100', 'ret_20_raw_norm_lag_101_120', 'ret_20_raw_norm_lag_121_140', 'ret_20_raw_norm_lag_141_160', 'ret_20_raw_norm_lag_161_180', 'ret_20_raw_norm_lag_181_200', 'ret_20_raw_norm_lag_201_220', 'ret_20_raw_norm_lag_221_240', 'ret_20_raw_norm_lag_241_260', 'ret_20_raw_norm_lag_261_280', 'isJan',

2.0.2 Model Output/Target

'target'

2.0.3 Logistic Regression - Baseline Strategy

Use 5 years data for training and subsequent 1 year data for testing

Before doing prediction with random forest, I prefer to see the correlation between features and also the hyper parameter for Random Forest need to be settled. And I will do 90/10 split of my training data from 20080214 to 20121231. Thus the training set will be from 20080214 to 20120630 an the test set be what is left.

2.0.4 Your Model

Use 5 years data for training and subsequent 1 year data for testing. For cross-validation, do 90/10 split of your training data to obtain optimal hyper-parameters.

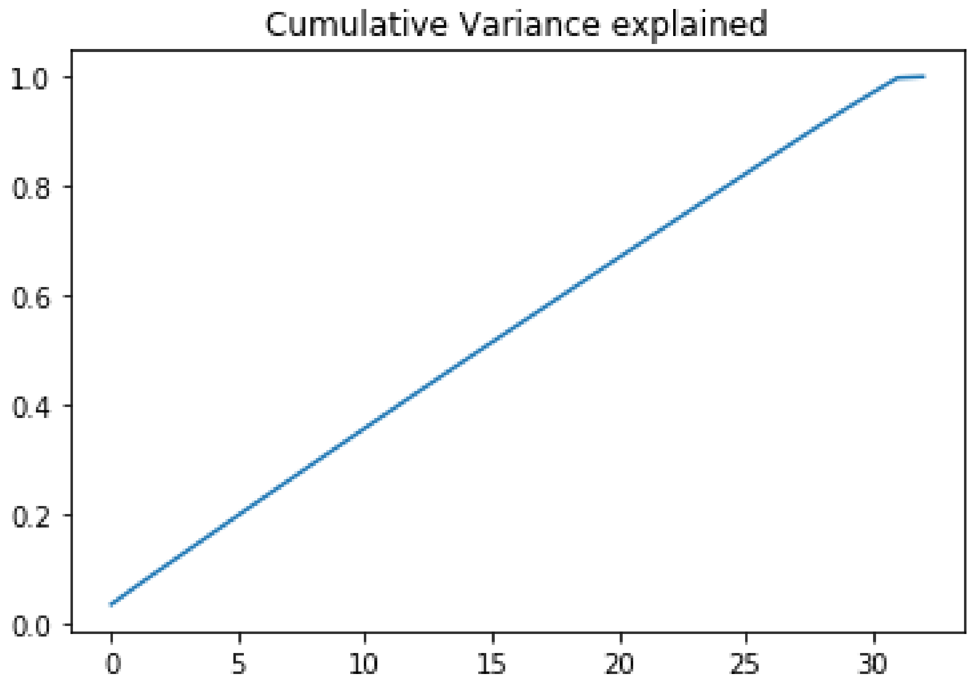
Before doing training, I want to have a look on the features first. I can't look on the whole dataset, so we will just look on the data set from 20080101 to 20120630.

```
In [7]: train=data[(data.date<=20120630)&(data.date>=20080101)]
        train=train.dropna()
        test=data[(data.date<=20121231)&(data.date>=20120631)]
        test=test.dropna()
        X_train=np.asarray(train.loc[:,'ret_raw_norm_lag_21':'isJan'])
        Y_train=np.asarray(train.loc[:,'target'])
        X_test=np.asarray(test.loc[:,'ret_raw_norm_lag_21':'isJan'])
        Y_test=np.asarray(test.loc[:,'target'])
```

```
In [8]: import numpy as np
        from sklearn.decomposition import PCA
        pca = PCA(copy=True, random_state=None,
          svd_solver='full', tol=0.0, whiten=False)
```

```
In [9]: X_train = pca.fit_transform(X_train)
```

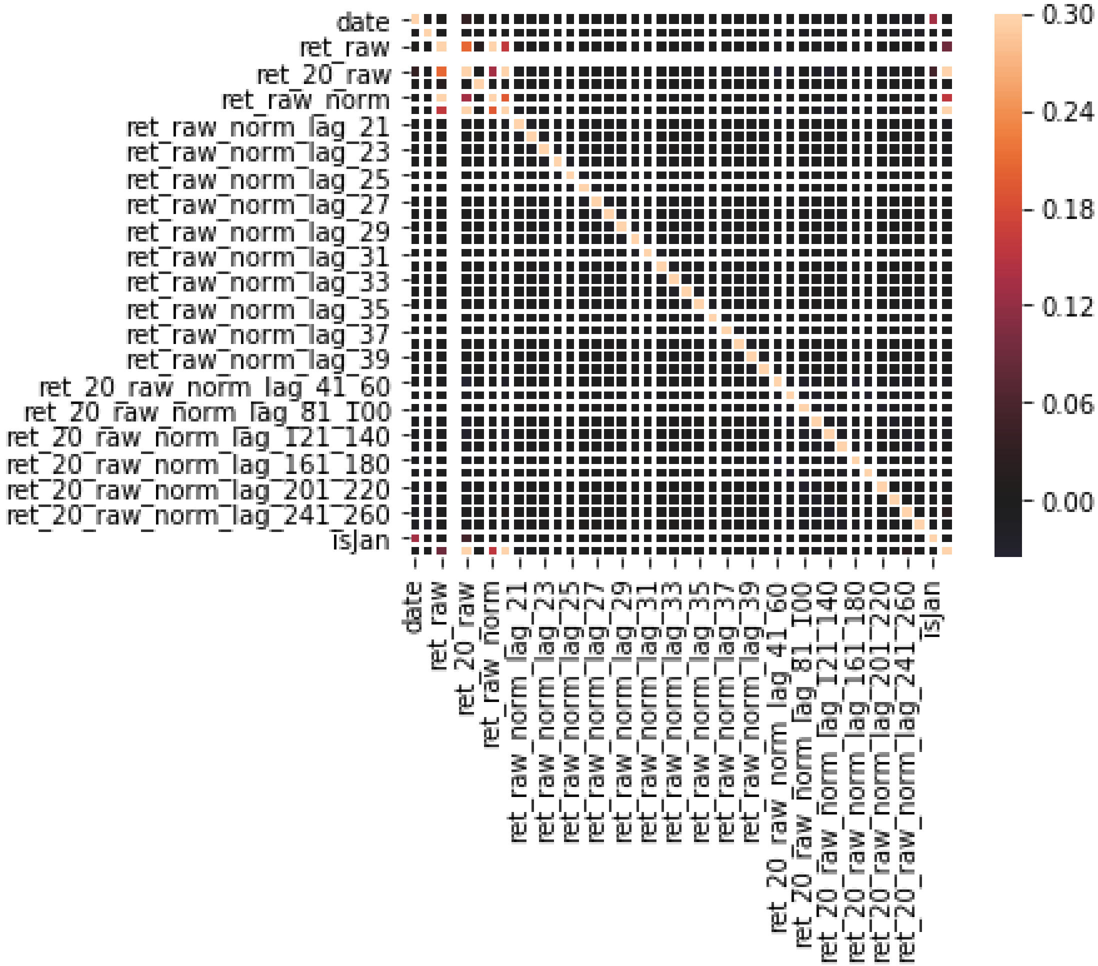
```
In [10]: _ = plt.plot(pca.explained_variance_ratio_.cumsum())
         _ = plt.title("Cumulative Variance explained")
```



We can see from the graph above that we need 22 features to reach 70% variance explained. The features are not correlated with each other. So the varaince explained ratio grows linearly. **So PCA here can't reduce correlation between features actually.** The features are not correlated. **PCA is not used here!**

```
In [11]: sns.heatmap(train.corr(), vmax=.3, center=0, square=True, linewidths=.5)
```

```
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x1f8a0da2630>
```



3 Training Procudure

Hyper parameter for Random Forest I do the cross validation on n_estimators, find 80 is a good value for that. However, after that I changed my code to calculate the grid search of {"max_depth" :[2,3,4,5],"max_leaf_nodes":[2,3,4,5]}. So the result of n_estimators is covered.(Sorry about that).

```
In [7]: from sklearn.ensemble import RandomForestClassifier
        from sklearn.model_selection import cross_val_score

        from sklearn.model_selection import GridSearchCV

        train=data[(data.date<=20120630)&(data.date>=20080101)]
        train=train.dropna()
        test=data[(data.date<=20121231)&(data.date>=20120631)]
        test=test.dropna()
        X_train=np.asarray(train.loc[:,'ret_raw_norm_lag_21':'isJan'])
```

```
Y_train=np.asarray(train.loc[:,'target'])
X_test=np.asarray(test.loc[:,'ret_raw_norm_lag_21':'isJan'])
Y_test=np.asarray(test.loc[:,'target'])

rf = RandomForestClassifier(n_estimators= 80)
parameters ={"max_depth" :[2,3,4,5],"max_leaf_nodes":[2,3,4,5]}
clf = GridSearchCV(rf, parameters, cv=10,n_jobs = 4)
clf.fit(X_train,Y_train)
```

Out[7]:
```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='g
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=80, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
       fit_params=None, iid='warn', n_jobs=4,
       param_grid={'max_depth': [2, 3, 4, 5], 'max_leaf_nodes': [2, 3, 4, 5]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

In [8]:
```
clf.cv_results_
```

```
C:\Users\olive\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
  warnings.warn(*warn_args, **warn_kwargs)
C:\Users\olive\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
  warnings.warn(*warn_args, **warn_kwargs)
C:\Users\olive\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
  warnings.warn(*warn_args, **warn_kwargs)
C:\Users\olive\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
  warnings.warn(*warn_args, **warn_kwargs)
C:\Users\olive\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
  warnings.warn(*warn_args, **warn_kwargs)
C:\Users\olive\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
  warnings.warn(*warn_args, **warn_kwargs)
C:\Users\olive\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
  warnings.warn(*warn_args, **warn_kwargs)
C:\Users\olive\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
  warnings.warn(*warn_args, **warn_kwargs)
C:\Users\olive\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
  warnings.warn(*warn_args, **warn_kwargs)
C:\Users\olive\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
  warnings.warn(*warn_args, **warn_kwargs)
C:\Users\olive\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
  warnings.warn(*warn_args, **warn_kwargs)
C:\Users\olive\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:125: FutureWarning: You
  warnings.warn(*warn_args, **warn_kwargs)
```

```
Out[8]: {'mean_fit_time': array([187.11656907, 232.45706127, 267.4149518 , 282.00506325,
                194.34117355, 249.25695157, 291.52386155, 302.00454996,
                192.84053221, 245.38647552, 281.41101382, 314.2137624 ,
                193.5640692 , 245.74409003, 278.23923907, 298.20747631]),
         'std_fit_time': array([ 9.04513028,  7.25578422,  3.77099695,  4.20912836,  1.6728912
                 2.95658279,  4.95447199,  3.54394342,  0.8174638 ,  2.16236446,
                 2.99722925,  7.76292081,  1.27714642,  4.07694091,  3.65530417,
                16.93318891]),
         'mean_score_time': array([0.98766005, 0.86429083, 0.95187697, 0.99464011, 0.91949608,
                0.91817837, 0.93061113, 0.93390419, 0.90725229, 0.91874504,
                0.94948773, 0.938293  , 0.94334202, 0.91266124, 0.90936904,
                0.88942559]),
         'std_score_time': array([0.15853096, 0.06853615, 0.04591754, 0.10148656, 0.04836237,
                0.04960147, 0.08448154, 0.02636992, 0.0271051 , 0.03701116,
                0.04396011, 0.02865862, 0.04251552, 0.03913344, 0.03737168,
                0.08048322]),
         'param_max_depth': masked_array(data=[2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5]
                      mask=[False, False, False, False, False, False, False, False,
                            False, False, False, False, False, False, False, False],
                fill_value='?',
                     dtype=object),
         'param_max_leaf_nodes': masked_array(data=[2, 3, 4, 5, 2, 3, 4, 5, 2, 3, 4, 5, 2, 3, 4
                      mask=[False, False, False, False, False, False, False, False,
                            False, False, False, False, False, False, False, False],
                fill_value='?',
                     dtype=object),
         'params': [{'max_depth': 2, 'max_leaf_nodes': 2},
          {'max_depth': 2, 'max_leaf_nodes': 3},
          {'max_depth': 2, 'max_leaf_nodes': 4},
          {'max_depth': 2, 'max_leaf_nodes': 5},
          {'max_depth': 3, 'max_leaf_nodes': 2},
          {'max_depth': 3, 'max_leaf_nodes': 3},
          {'max_depth': 3, 'max_leaf_nodes': 4},
          {'max_depth': 3, 'max_leaf_nodes': 5},
          {'max_depth': 4, 'max_leaf_nodes': 2},
          {'max_depth': 4, 'max_leaf_nodes': 3},
          {'max_depth': 4, 'max_leaf_nodes': 4},
          {'max_depth': 4, 'max_leaf_nodes': 5},
          {'max_depth': 5, 'max_leaf_nodes': 2},
          {'max_depth': 5, 'max_leaf_nodes': 3},
          {'max_depth': 5, 'max_leaf_nodes': 4},
          {'max_depth': 5, 'max_leaf_nodes': 5}],
         'split0_test_score': array([0.49504505, 0.49416617, 0.49901731, 0.49984775, 0.5046504
                0.51022823, 0.49051916, 0.49528726, 0.50570926, 0.49502429,
                0.50302418, 0.50188925, 0.48729429, 0.50276121, 0.50116261,
                0.50232523]),
         'split1_test_score': array([0.49507273, 0.49757789, 0.49833912, 0.49830452, 0.4952042
                0.49851905, 0.49765401, 0.49968859, 0.49519038, 0.50198613,
```

```
       0.49855365, 0.49916264, 0.49893427, 0.49665749, 0.49839449,
       0.49968859]),
'split2_test_score': array([0.51435966, 0.51397905, 0.51716239, 0.52119002, 0.51928001
       0.51716239, 0.51960526, 0.51959142, 0.52105853, 0.5164842 ,
       0.5195153 , 0.52009661, 0.51647036, 0.5252038 , 0.52099625,
       0.51485793]),
'split3_test_score': array([0.49334607, 0.49089626, 0.49041183, 0.49068865, 0.49326994
       0.48677172, 0.49314538, 0.49020422, 0.49034955, 0.48711082,
       0.48837032, 0.4920312 , 0.48868174, 0.49079937, 0.48993433,
       0.49072325]),
'split4_test_score': array([0.5052491 , 0.50935288, 0.50470931, 0.51002415, 0.50345672
       0.51135978, 0.50630791, 0.51084076, 0.50264012, 0.5079688 ,
       0.50553975, 0.50867468, 0.5067439 , 0.50762278, 0.50913835,
       0.50310378]),
'split5_test_score': array([0.50519031, 0.50517647, 0.50611765, 0.50436678, 0.50530104
       0.50519723, 0.50407612, 0.50489273, 0.50497578, 0.50487197,
       0.50643599, 0.50442215, 0.50602768, 0.50547405, 0.5053564 ,
       0.50521799]),
'split6_test_score': array([0.49470588, 0.49266436, 0.49777855, 0.49957785, 0.49742561
       0.49515571, 0.49541869, 0.496609  , 0.49466436, 0.49678893,
       0.49959862, 0.49608304, 0.49986159, 0.49651211, 0.4944083 ,
       0.49956401]),
'split7_test_score': array([0.51658824, 0.51463668, 0.52136332, 0.5103391 , 0.5153564
       0.51575779, 0.51352941, 0.5150519 , 0.52383391, 0.50991696,
       0.51887197, 0.51580623, 0.50062976, 0.51244983, 0.51360554,
       0.51674048]),
'split8_test_score': array([0.52422837, 0.52112111, 0.52141176, 0.52197232, 0.52405536
       0.52190311, 0.52083045, 0.51965398, 0.52133564, 0.52060208,
       0.52281661, 0.51946713, 0.51640138, 0.5190519 , 0.51957093,
       0.52283737]),
'split9_test_score': array([0.49683737, 0.49519723, 0.49725952, 0.50011765, 0.49833218
       0.49494118, 0.49894118, 0.49461592, 0.49484429, 0.49686505,
       0.49555017, 0.4929827 , 0.50373702, 0.49277509, 0.49829066,
       0.49118339]),
'mean_test_score': array([0.50406226, 0.5034768 , 0.50535706, 0.50564288, 0.50563319,
       0.50569962, 0.50400275, 0.50464357, 0.50546018, 0.50376192,
       0.50582765, 0.50506156, 0.50247819, 0.50493077, 0.50508578,
       0.5046242 ]),
'std_test_score': array([0.01043546, 0.01024106, 0.0104409 , 0.00964168, 0.01004116,
       0.01088419, 0.01030529, 0.0103983 , 0.01185001, 0.00977349,
       0.01075425, 0.01001822, 0.00930448, 0.010798  , 0.01000979,
       0.01007501]),
'rank_test_score': array([12, 15,  6,  3,  4,  2, 13, 10,  5, 14,  1,  8, 16,  9,  7,
'split0_train_score': array([0.51892879, 0.52021598, 0.51937554, 0.52245741, 0.5185481
       0.51744244, 0.52062736, 0.52268579, 0.51909488, 0.52088418,
       0.51988918, 0.52167772, 0.51652049, 0.52039206, 0.52161389,
       0.5223836 ]),
'split1_train_score': array([0.51904874, 0.51987688, 0.51981998, 0.52229824, 0.51830013
```

```
        0.52012524, 0.52089033, 0.52104719, 0.51914101, 0.52050663,
        0.521141  , 0.52185842, 0.51947934, 0.51819753, 0.52161313,
        0.52233977]),
 'split2_train_score': array([0.51453434, 0.51487575, 0.51709565, 0.51784536, 0.5163290
        0.51535172, 0.51773925, 0.51707028, 0.51533788, 0.51646974,
        0.51760238, 0.51879115, 0.51486729, 0.51565929, 0.51774309,
        0.51726482]),
 'split3_train_score': array([0.52050162, 0.52046317, 0.52090608, 0.52201257, 0.5203770
        0.51903219, 0.52192337, 0.52256005, 0.51935207, 0.51884534,
        0.52052238, 0.52344432, 0.51745512, 0.52127593, 0.52110754,
        0.52232245]),
 'split4_train_score': array([0.51815638, 0.51958352, 0.51959582, 0.52074076, 0.5184370
        0.5189653 , 0.51974192, 0.52109985, 0.51766965, 0.51946433,
        0.51952892, 0.52014714, 0.51851547, 0.51924519, 0.52017713,
        0.51963119]),
 'split5_train_score': array([0.51796144, 0.51957081, 0.51895721, 0.52110791, 0.5173578
        0.51915405, 0.52052736, 0.51970845, 0.51633285, 0.51767848,
        0.52026054, 0.5210487 , 0.51896797, 0.51927478, 0.52059426,
        0.52079649]),
 'split6_train_score': array([0.5197346 , 0.51971999, 0.52040741, 0.51985455, 0.5192916
        0.52025593, 0.520665  , 0.52079188, 0.52064578, 0.51995912,
        0.5197938 , 0.5215139 , 0.51981918, 0.51979534, 0.52111944,
        0.52023286]),
 'split7_train_score': array([0.51729786, 0.51859812, 0.51923018, 0.52079418, 0.5178407
        0.51873806, 0.52029745, 0.52062502, 0.5162698 , 0.51988915,
        0.51986301, 0.52099949, 0.51771539, 0.5191479 , 0.52023901,
        0.52086415]),
 'split8_train_score': array([0.51629517, 0.51711408, 0.51952698, 0.5207765 , 0.5159653
        0.51690109, 0.51782611, 0.51896643, 0.51541013, 0.51738859,
        0.52000449, 0.52065808, 0.51418369, 0.51637591, 0.51751239,
        0.52101102]),
 'split9_train_score': array([0.51894875, 0.51962541, 0.52078726, 0.52144931, 0.5175454
        0.5199276 , 0.52024978, 0.52122863, 0.51904333, 0.51998834,
        0.5206404 , 0.52159695, 0.51847047, 0.51985224, 0.52110483,
        0.5220014 ]),
 'mean_train_score': array([0.51814077, 0.51896437, 0.51957021, 0.52093368, 0.51799936
        0.51858936, 0.52004879, 0.52057836, 0.51782974, 0.51910739,
        0.51992461, 0.52117359, 0.51759944, 0.51892162, 0.52028247,
        0.52088477]),
 'std_train_score': array([0.00165431, 0.00163338, 0.00103365, 0.0012817 , 0.00124409,
        0.00149177, 0.00125046, 0.0015809 , 0.00178337, 0.00139201,
        0.00089666, 0.00114959, 0.00179617, 0.00165115, 0.00140829,
        0.00151003])}
```

In [9]:
```
clf.best_estimator_
```

Out[9]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=4, max_features='auto', max_leaf_nodes=4,
```

```
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators=80, n_jobs=None,
                    oob_score=False, random_state=None, verbose=0,
                    warm_start=False)
```

```
In [10]: clf.best_score_
```

```
Out[10]: 0.5058276493970968
```

Hyper parameter and model selection for Dnn

```
In [3]: # to make this notebook's output stable across runs
        def reset_graph(seed=42):
            tf.reset_default_graph()
            tf.set_random_seed(seed)
            np.random.seed(seed)

        # To plot pretty figures
        %matplotlib inline
        import matplotlib
        import matplotlib.pyplot as plt
        plt.rcParams['axes.labelsize'] = 14
        plt.rcParams['xtick.labelsize'] = 12
        plt.rcParams['ytick.labelsize'] = 12
```

```
In [4]: from functools import partial
```

```
In [5]: reset_graph()

        n_inputs = 33
        n_hidden1 = 50
        n_hidden2 = 30
        n_hidden3 = 10
        n_outputs = 2
```

```
In [6]: X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")
        y = tf.placeholder(tf.int32, shape=(None), name="y")
        he_init = tf.variance_scaling_initializer()
```

```
In [7]: training = tf.placeholder_with_default(False, shape=(), name='training')

        dropout_rate = 0.8  # == 1 - keep_prob
        X_drop = tf.layers.dropout(X, dropout_rate, training=training)

        with tf.name_scope("MLP"):
            hidden1 = tf.layers.dense(X, n_hidden1, activation=tf.nn.elu,kernel_initializer=he_
            hidden1_drop = tf.layers.dropout(hidden1, dropout_rate, training=training)
```

```
        hidden2 = tf.layers.dense(hidden1_drop, n_hidden2, activation=tf.nn.elu,kernel_ini
        hidden2_drop = tf.layers.dropout(hidden2, dropout_rate, training=training)

        hidden3 = tf.layers.dense(hidden2_drop, n_hidden3, activation=tf.nn.elu,kernel_ini



        logits = tf.layers.dense(hidden3, n_outputs, name="outputs")
```

```
In [8]: with tf.name_scope("loss"):
            xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y, logits=logits)
            loss = tf.reduce_mean(xentropy, name="loss")
```

```
In [12]: learning_rate = 0.001

         with tf.name_scope("train"):
             optimizer = tf.train.AdamOptimizer(learning_rate)
             training_op = optimizer.minimize(loss)
```

```
In [13]: with tf.name_scope("eval"):
             correct = tf.nn.in_top_k(logits, y, 1)
             accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
```

```
In [14]: saver = tf.train.Saver()
         init = tf.global_variables_initializer()

         prediction = tf.nn.softmax(logits)
```

```
In [15]: def shuffle_batch(X, y, batch_size):
             rnd_idx = np.random.permutation(len(X))
             n_batches = batch_size
             for batch_idx in np.array_split(rnd_idx, n_batches):
                 X_batch, y_batch = X[batch_idx], y[batch_idx]
                 yield X_batch, y_batch
```

```
In [137]: train=data[(data.date<=20120630)&(data.date>=20080101)]
          train=train.dropna()
          test=data[(data.date<=20121231)&(data.date>=20120631)]
          test=test.dropna()
          X_train=np.asarray(train.loc[:,'ret_raw_norm_lag_21':'isJan'])
          Y_train=np.asarray(train.loc[:,'target'])
          X_test=np.asarray(test.loc[:,'ret_raw_norm_lag_21':'isJan'])
          Y_test=np.asarray(test.loc[:,'target'])
```

```
In [138]: X_train.shape
```

```
Out[138]: (1445008, 33)
```

```
In [171]: n_epochs = 80
          batch_size = 256
          with tf.Session() as sess:
              init.run()
              for epoch in range(n_epochs):
                  for X_batch, y_batch in shuffle_batch(X_train, Y_train, batch_size):
                      sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
                  if epoch % 5 == 0:
                      acc_batch = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
                      acc_valid = accuracy.eval(feed_dict={X: X_test, y: Y_test})
                      print(epoch, "Batch accuracy:", acc_batch, "Validation accuracy:", acc_va

              save_path = saver.save(sess, "./train01/my_model_final.ckpt")
              Predict_prob = sess.run(prediction, feed_dict={X: X_test})
```

```
0 Batch accuracy: 0.51947564 Validation accuracy: 0.51542544
5 Batch accuracy: 0.52709115 Validation accuracy: 0.5135109
10 Batch accuracy: 0.525593 Validation accuracy: 0.51395005
15 Batch accuracy: 0.5344569 Validation accuracy: 0.51295227
20 Batch accuracy: 0.5202247 Validation accuracy: 0.5125685
25 Batch accuracy: 0.53533083 Validation accuracy: 0.5111485
30 Batch accuracy: 0.5300874 Validation accuracy: 0.50935763
35 Batch accuracy: 0.54194754 Validation accuracy: 0.50885016
40 Batch accuracy: 0.5382022 Validation accuracy: 0.50922114
45 Batch accuracy: 0.52821475 Validation accuracy: 0.5064665
50 Batch accuracy: 0.53395754 Validation accuracy: 0.5080826
55 Batch accuracy: 0.5354557 Validation accuracy: 0.5046415
60 Batch accuracy: 0.5445693 Validation accuracy: 0.50559664
65 Batch accuracy: 0.55081147 Validation accuracy: 0.5046159
70 Batch accuracy: 0.5420724 Validation accuracy: 0.50459886
75 Batch accuracy: 0.5385768 Validation accuracy: 0.5027653
```

Cross validation for DNN

```
In [204]: train=data[(data.date<=20121231)&(data.date>=20080101)]
          train=train.dropna()

          X_train=np.asarray(train.loc[:,'ret_raw_norm_lag_21':'isJan'])
          Y_train=np.asarray(train.loc[:,'target'])
```

```
In [136]: X.shape
```

```
Out[136]: (1445008, 33)
```

```
In [156]: from sklearn.model_selection import train_test_split
          for i in np.arange(0,10):
              X_train_1, X_test_1, y_train_1, y_test_1 = train_test_split(X_train, Y_train, tes
              n_epochs = 70
```

```
        batch_size = 256
        with tf.Session() as sess:
            init.run()
            for epoch in range(n_epochs):
                for X_batch, y_batch in shuffle_batch(X_train_1, y_train_1, batch_size):
                    sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
                if epoch % 5 == 0:
                    acc_batch = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
                    acc_valid = accuracy.eval(feed_dict={X: X_test_1, y: y_test_1})
                    if epoch == n_epochs-5:
                        print("{}th time split:".format(i+1))
                        print(epoch, "Batch accuracy:", acc_batch, "Validation accuracy:'

            save_path = saver.save(sess, "./train01/my_model_final.ckpt")
            Predict_prob = sess.run(prediction, feed_dict={X: X_test})
```

```
1th time split:
65 Batch accuracy: 0.55111665 Validation accuracy: 0.53406286
2th time split:
65 Batch accuracy: 0.5434873 Validation accuracy: 0.53260475
3th time split:
65 Batch accuracy: 0.5437647 Validation accuracy: 0.534453
4th time split:
65 Batch accuracy: 0.5389097 Validation accuracy: 0.53584284
5th time split:
65 Batch accuracy: 0.54015815 Validation accuracy: 0.5355697
6th time split:
65 Batch accuracy: 0.54390347 Validation accuracy: 0.53306806
7th time split:
65 Batch accuracy: 0.5448745 Validation accuracy: 0.535204
8th time split:
65 Batch accuracy: 0.54404217 Validation accuracy: 0.53662795
9th time split:
65 Batch accuracy: 0.5380774 Validation accuracy: 0.5347163
10th time split:
65 Batch accuracy: 0.5412679 Validation accuracy: 0.5333899
```

4 Doing test and generating result

Predictions for Random Forest Making predictions on Test with rolling window

```
In [16]: from sklearn.ensemble import RandomForestClassifier
         from sklearn.model_selection import cross_val_score
         for i in range (2012,2013):
             train=data[(data.date<=int(str(i)+'1231'))&(data.date>=int(str(i-4)+'0101'))]
             train=train.dropna()
             train.index=range(len(train))
```

```
        #globals()['train_{}'.format(i)]=train.copy()

        test=data[(data.date<=int(str(i+1)+'1231'))&(data.date>=int(str(i+1)+'0101'))]
        test=test.dropna()
        test.index=range(len(test))
        #globals()['test_{}'.format(i)]=test.copy()

        X_train=np.asarray(train.loc[:,'ret_raw_norm_lag_21':'isJan'])
        Y_train=np.asarray(train.loc[:,'target'])
        X_test=np.asarray(test.loc[:,'ret_raw_norm_lag_21':'isJan'])
        Y_test=np.asarray(test.loc[:,'target'])

        #Using Cross validation to find suitbale hyper parameter

    #  score = []
    #  parameter_list = [20,50,80,100]
    #  for item in parameter_list:
    #      rf = RandomForestClassifier(n_estimators =item,n_jobs=4)
    #      score.append(np.mean(cross_val_score(rf, X_train, Y_train, cv=10)))
    #      print(score)
    #  parameter_picked = parameter_list[score.index(np.max(score))]



        clf= RandomForestClassifier(n_jobs=4,n_estimators = 80,max_leaf_nodes=4,max_depth
        clf.fit(X_train, Y_train)
        #res=pd.concat((test.loc[:,['id','date','target']], pd.DataFrame(clf.predict_prob
        res=pd.concat((train.loc[:,['id','date','target']], pd.DataFrame(clf.predict_proba
        res.columns=["id","date",'target',"pred_zsprob_comp","Alp"]
        res.index=range(len(res))
        globals()["result_{}_rf".format(i)]=res.copy()
        print("result_{}_rf".format(i))
```

```
result_2012_rf
```

```
In [17]: all_score=[]
         for i in range (2012,2017):
             train=data[(data.date<=int(str(i)+'1231'))&(data.date>=int(str(i-4)+'0101'))]
             train=train.dropna()
             train.index=range(len(train))
             #globals()['train_{}'.format(i)]=train.copy()

             test=data[(data.date<=int(str(i+1)+'1231'))&(data.date>=int(str(i+1)+'0101'))]
             test=test.dropna()
             test.index=range(len(test))
             #globals()['test_{}'.format(i)]=test.copy()
```

```
        X_train=np.asarray(train.loc[:,'ret_raw_norm_lag_21':'isJan'])
        Y_train=np.asarray(train.loc[:,'target'])

        X_test=np.asarray(test.loc[:,'ret_raw_norm_lag_21':'isJan'])
        Y_test=np.asarray(test.loc[:,'target'])

        #ueing Cross validation to find suitbale hyper parameter
    #    score = []
    #    parameter_list = [20,50,80,100]
     ##   for item in parameter_list:
     #       rf = RandomForestClassifier(n_estimators =item,n_jobs=-1)
     #       score.append(np.mean(cross_val_score(rf, X_train, Y_train, cv=10)))
    #    parameter_picked = parameter_list[score.index(np.max(score))]
       # all_score.append(score)
    #

        clf= RandomForestClassifier(n_estimators =80,n_jobs=4,max_leaf_nodes=4,max_depth =
        clf.fit(X_train, Y_train)

        res=pd.concat((test.loc[:,['id','date','target']], pd.DataFrame(clf.predict_proba
        #res=pd.concat((train.loc[:,['id','date','target']], pd.DataFrame(clf.predict_pro
        res.columns=["id","date",'target',"pred_zsprob_comp","Alp"]
        res.index=range(len(res))
        globals()["result_{}_rf".format(i+1)]=res.copy()
        print("result_{}_rf".format(i+1))
```

```
result_2013_rf
result_2014_rf
result_2015_rf
result_2016_rf
result_2017_rf
```

```
In [18]: RES=result_2012_rf
         for i in range(2013,2018):
             RES=pd.concat((RES,globals()["result_{}_rf".format(i)]),axis=0)

         tt_rf=RES.merge(data.loc[:,["id","date","ret_raw","ret_20_raw","industry","flag2"]],le
```

Predictions for Logistic Regression

```
In [19]: for i in range (2012,2013):
             train=data[(data.date<=int(str(i)+'1231'))&(data.date>=int(str(i-4)+'0101'))]
             train=train.dropna()
             train.index=range(len(train))
             #globals()['train_{}'.format(i)]=train.copy()

             test=data[(data.date<=int(str(i+1)+'1231'))&(data.date>=int(str(i+1)+'0101'))]
```

```
            test=test.dropna()
            test.index=range(len(test))
            #globals()['test_{}'.format(i)]=test.copy()

            X_train=np.asarray(train.loc[:,'ret_raw_norm_lag_21':'isJan'])
            Y_train=np.asarray(train.loc[:,'target'])

            X_test=np.asarray(test.loc[:,'ret_raw_norm_lag_21':'isJan'])
            Y_test=np.asarray(test.loc[:,'target'])

            clf=LogisticRegression(n_jobs=-1, solver='lbfgs')
            clf.fit(X_train, Y_train)
            #res=pd.concat((test.loc[:,['id','date','target']], pd.DataFrame(clf.predict_prob
            res=pd.concat((train.loc[:,['id','date','target']], pd.DataFrame(clf.predict_proba
            res.columns=["id","date",'target',"pred_zsprob_comp","Alp"]
            res.index=range(len(res))
            globals()["result_{}_logistic".format(i)]=res.copy()
            print("result_{}_logistic".format(i))

result_2012_logistic
```

```
In [20]: for i in range (2012,2017):
            train=data[(data.date<=int(str(i)+'1231'))&(data.date>=int(str(i-4)+'0101'))]
            train=train.dropna()
            train.index=range(len(train))
            #globals()['train_{}'.format(i)]=train.copy()

            test=data[(data.date<=int(str(i+1)+'1231'))&(data.date>=int(str(i+1)+'0101'))]
            test=test.dropna()
            test.index=range(len(test))
            #globals()['test_{}'.format(i)]=test.copy()

            X_train=np.asarray(train.loc[:,'ret_raw_norm_lag_21':'isJan'])
            Y_train=np.asarray(train.loc[:,'target'])

            X_test=np.asarray(test.loc[:,'ret_raw_norm_lag_21':'isJan'])
            Y_test=np.asarray(test.loc[:,'target'])

            clf=LogisticRegression(n_jobs=-1, solver='lbfgs')
            clf.fit(X_train, Y_train)
            res=pd.concat((test.loc[:,['id','date','target']], pd.DataFrame(clf.predict_proba
            #res=pd.concat((train.loc[:,['id','date','target']], pd.DataFrame(clf.predict_pro
            res.columns=["id","date",'target',"pred_zsprob_comp","Alp"]
            res.index=range(len(res))
            globals()["result_{}_logistic".format(i+1)]=res.copy()
            print("result_{}_logistic".format(i+1))

result_2013_logistic
```

```
result_2014_logistic
result_2015_logistic
result_2016_logistic
result_2017_logistic
```

```
In [21]: RES=result_2012_logistic
         for i in range(2013,2018):
             RES=pd.concat((RES,globals()["result_{}_logistic".format(i)]),axis=0)

         tt_logistic=RES.merge(data.loc[:,["id","date","ret_raw","ret_20_raw","industry","flag2
```

Predictions for DNN

```
In [22]: for i in range (2012,2013):
             train=data[(data.date<=int(str(i)+'1231'))&(data.date>=int(str(i-4)+'0101'))]
             train=train.dropna()
             train.index=range(len(train))
             #globals()['train_{}'.format(i)]=train.copy()

             test=data[(data.date<=int(str(i+1)+'1231'))&(data.date>=int(str(i+1)+'0101'))]
             test=test.dropna()
             test.index=range(len(test))
             #globals()['test_{}'.format(i)]=test.copy()

             X_train=np.asarray(train.loc[:,'ret_raw_norm_lag_21':'isJan'])
             Y_train=np.asarray(train.loc[:,'target'])

             X_test=np.asarray(test.loc[:,'ret_raw_norm_lag_21':'isJan'])
             Y_test=np.asarray(test.loc[:,'target'])


             n_epochs = 40
             batch_size = 256
             with tf.Session() as sess:
                 init.run()
                 for epoch in range(n_epochs):
                     for X_batch, y_batch in shuffle_batch(X_train, Y_train, batch_size):
                         sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
                     if epoch % 5 == 0:
                         acc_batch = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
                         acc_valid = accuracy.eval(feed_dict={X: X_test, y: Y_test})
                         if epoch == n_epochs-5:
                             print("{}th time split:".format(i))
                             print(epoch, "Batch accuracy:", acc_batch, "Validation accuracy:"

                 save_path = saver.save(sess, "./train01/my_model_final.ckpt")
                 Predict_prob = sess.run(prediction, feed_dict={X: X_train})
```

```
        #res=pd.concat((test.loc[:,['id','date','target']], pd.DataFrame(clf.predict_prob
        res=pd.concat((train.loc[:,['id','date','target']], pd.DataFrame(Predict_prob)),a
        res.columns=["id","date",'target',"pred_zsprob_comp","Alp"]
        res.index=range(len(res))
        globals()["result_{}_dnn".format(i)]=res.copy()
        print("result_{}_dnn".format(i))
```

```
2012th time split:
35 Batch accuracy: 0.5381236 Validation accuracy: 0.4999796
result_2012_dnn
```

```
In [23]: for i in range (2012,2017):
             train=data[(data.date<=int(str(i)+'1231'))&(data.date>=int(str(i-4)+'0101'))]
             train=train.dropna()
             train.index=range(len(train))
             #globals()['train_{}'.format(i)]=train.copy()

             test=data[(data.date<=int(str(i+1)+'1231'))&(data.date>=int(str(i+1)+'0101'))]
             test=test.dropna()
             test.index=range(len(test))
             #globals()['test_{}'.format(i)]=test.copy()

             X_train=np.asarray(train.loc[:,'ret_raw_norm_lag_21':'isJan'])
             Y_train=np.asarray(train.loc[:,'target'])

             X_test=np.asarray(test.loc[:,'ret_raw_norm_lag_21':'isJan'])
             Y_test=np.asarray(test.loc[:,'target'])


             n_epochs = 40
             batch_size = 256
             with tf.Session() as sess:
                 init.run()
                 for epoch in range(n_epochs):
                     for X_batch, y_batch in shuffle_batch(X_train, Y_train, batch_size):
                         sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
                     if epoch % 5 == 0:
                         acc_batch = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
                         acc_valid = accuracy.eval(feed_dict={X: X_test, y: Y_test})
                         if epoch == n_epochs-5:
                             print("{}th time split:".format(i))
                             print(epoch, "Batch accuracy:", acc_batch, "Validation accuracy:"
```

```
            save_path = saver.save(sess, "./train01/my_model_final.ckpt")
            Predict_prob = sess.run(prediction, feed_dict={X: X_test})

        res=pd.concat((test.loc[:,['id','date','target']], pd.DataFrame(Predict_prob)),axi
        #res=pd.concat((train.loc[:,['id','date','target']], pd.DataFrame(clf.predict_pro
        res.columns=["id","date",'target',"pred_zsprob_comp","Alp"]
        res.index=range(len(res))
        globals()["result_{}_dnn".format(i+1)]=res.copy()
        print("result_{}_dnn".format(i+1))
```

```
2012th time split:
35 Batch accuracy: 0.54753685 Validation accuracy: 0.49979848
result_2013_dnn
2013th time split:
35 Batch accuracy: 0.53758407 Validation accuracy: 0.50366896
result_2014_dnn
2014th time split:
35 Batch accuracy: 0.53927636 Validation accuracy: 0.51346695
result_2015_dnn
2015th time split:
35 Batch accuracy: 0.544815 Validation accuracy: 0.49954534
result_2016_dnn
2016th time split:
35 Batch accuracy: 0.53370786 Validation accuracy: 0.5039635
result_2017_dnn
```

The accuracy above should be `test accuracy`

```
In [24]: RES=result_2012_dnn
         for i in range(2013,2018):
             RES=pd.concat((RES,globals()["result_{}_dnn".format(i)]),axis=0)
         tt_dnn = 0
         tt_dnn=RES.merge(data.loc[:,["id","date","ret_raw","ret_20_raw","industry","flag2"]],
```

4.0.1 Back Test Framework

To use this back-test framework, input a dataframe with date, alpha, ret=ret_20_raw, flag as your holding period, and keep the quantile cut at q=0.9.

```
In [25]: def back_test(data_all,q=0.9,Alp="Alpha",flag="flag",date="date",ret="ret_raw",indust
             res_all = []
             for i,data in enumerate(data_all):

                 data=data[data.id!='GGP US Equity']
                 data=data[data.industry!=7777]
                 data=data[data.loc[:,flag]==True]
                 data=data.loc[:,[date,Id,Alp,ret,industry]]
                 data.columns=["Date","Id","Alp","Return","industry"]
```

```
            data.index=range(len(data))
            res=data.groupby(("Date")).apply(lambda x: x[x["Alp"]>=x["Alp"].quantile(q)].I
                                    x[x["Alp"]<=x["Alp"].quantile(1-q)].Return.mean()*0.5)
            positions=data.groupby(("Date")).apply(lambda x: x[(x["Alp"]>=x["Alp"].quanti

            long=data.groupby(("Date")).apply(lambda x: x[(x["Alp"]>=x["Alp"].quantile(q)
            short=data.groupby(("Date")).apply(lambda x: x[(x["Alp"]<=x["Alp"].quantile(1-
            res=res.reset_index()
            res=res.dropna()
            res.index=range(len(res))

            globals()["res{}".format(i)]=res.copy()
            res_all.append(res)
        return long,short,res_all
```

Compute Yearly Return and Sharpe Ratio

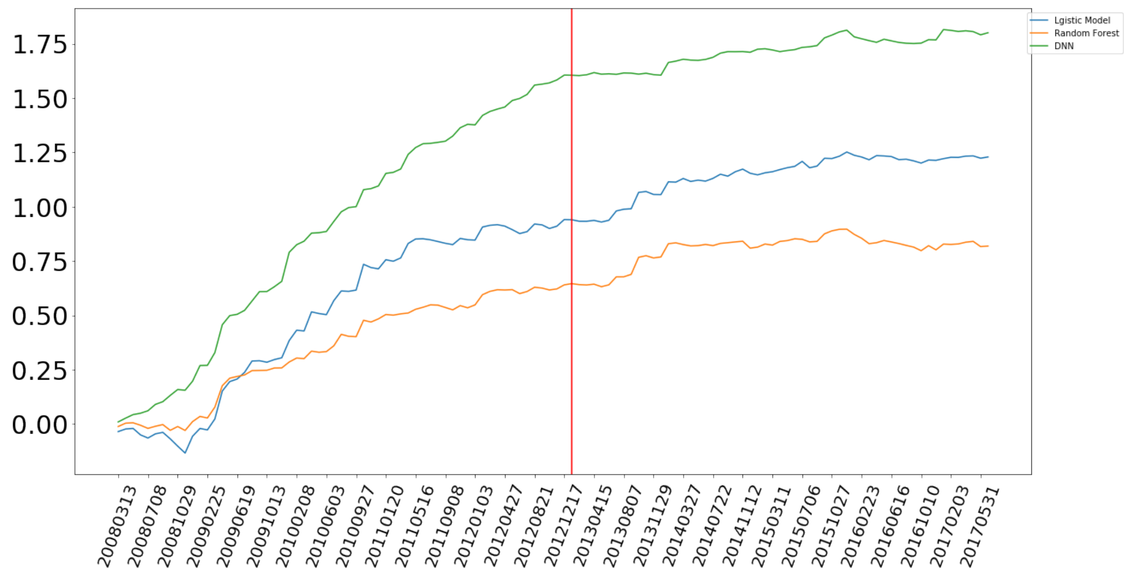
```
In [26]: def summary_return_sharpe(res,period):
        res.loc[:,"year"]=(res.Date/10000).apply(int)
        Ret=res.groupby("year")[0].sum().reset_index()
        Y10T=0.021
        n=int(253/period)
        Fday=(Y10T+1)**(1/n)-1
        Sharpe=res.groupby("year")[0].apply(lambda x: (x).mean()/((x).std())*np.sqrt(n)).r
        Performance=Ret.merge(Sharpe,left_on="year",right_on="year",how="left")
        Performance.columns=["Year","Return","Sharpe"]
        return Performance
```

4.0.2 Result

```
In [27]: data_all = [tt_logistic,tt_rf,tt_dnn]
         a3=back_test(data_all,q=0.90,flag="flag2",ret="ret_20_raw",Alp="Alp")

In [29]: plt.figure(figsize=(20,10))
         plt.plot((res0.iloc[:,-1]).cumsum())
         plt.plot((res1.iloc[:,-1]).cumsum())
         plt.plot((res2.iloc[:,-1]).cumsum())
         n=int(len(res0)/30)+1
         plt.legend(['Lgistic Model','Random Forest','DNN'],bbox_to_anchor=(1.1,1))
         plt.xticks(range(0,len(res0),n),
                        [str(res0.loc[i,"Date"]).split(" ")[0] for i in range(0,len(res0),n)],
                        rotation=70,
                       fontsize = 20)
         plt.yticks(fontsize=30)
         plt.axvline(x=len(res0[res0.Date<20121231]), c="r")

         plt.show()
```




```
In [31]: m0 = summary_return_sharpe(a3[2][0],12)
         m1 = summary_return_sharpe(a3[2][1],12)
         m2 = summary_return_sharpe(a3[2][2],12)

In [34]: Sharpe_ratio = pd.DataFrame({"Sharpe_logit":m0["Sharpe"],"Sharpe_rf":m1["Sharpe"],"Sha
         Sharpe_ratio.index = m0["Year"]
         Sharpe_ratio
```

Out[34]:

Year	Sharpe_logit	Sharpe_rf	Sharpe_Dnn
2008	-0.692299	0.226218	6.450617
2009	3.676245	3.199213	4.851014
2010	3.224571	3.088588	4.148105
2011	2.118230	1.675019	5.065845
2012	1.429364	2.323503	5.634900
2013	1.680922	1.898413	-0.033996
2014	1.818474	0.731912	2.377666
2015	2.223340	2.642948	3.339877
2016	-1.367387	-2.509084	-1.432808
2017	1.587530	0.772120	1.062946

```
In [33]: Return= pd.DataFrame({"Return_logit":m0["Return"],"Return_rf":m1["Return"],"Return_Dnn
         Return.index = m0["Year"]
         Return
```

Out[33]:

Year	Return_logit	Return_rf	Return_Dnn
2008	-0.056674	0.010669	0.196686
2009	0.361200	0.247267	0.459812

2010	0.409331	0.225744	0.439035
2011	0.134093	0.050870	0.283906
2012	0.092953	0.105688	0.226385
2013	0.114847	0.128510	-0.000489
2014	0.098261	0.040437	0.105563
2015	0.097750	0.087398	0.101748
2016	-0.038390	-0.095150	-0.044870
2017	0.015641	0.017558	0.033002