Limit order book

author: Jian Wang

time: 2017-06-17

Model training and fitting

1.Model prepare

In [1]:

```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 26 00:03:47 2016

@author: jianwang
"""

import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy as sp
from sklearn import linear_model
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import svm
from sklearn import tree
from sklearn import ensemble
import time
import matplotlib.pyplot as plt

#Set default parameters
ticker_list=["AAPL","AMZN","GOOG","INTC","MSFT"]
start_ind=10*3600
end_ind=15.5*3600
data_order_list=[]
data_mess_list=[]
time_index_list=[]
path_save='/media/jianwang/Study1/Research/order_book/'
path_load="/media/jianwang/Study1/Research/order_book/"

## set random seed to produce the same results
```

```python
np.random.seed(987612345)

#read the stock ticker
#totally 5 dataset

for i in range(len(ticker_list)):
    #get the path for the csv files
    # name_order is for the order book and name_mess for the message book
    name_order='_2012-06-21_34200000_57600000_orderbook_10.csv'
    name_mess='_2012-06-21_34200000_57600000_message_10.csv'
    # calculate the cputime for reading the data
    t=time.time()
    # header =-1 means that the first line is not the header, otherwise, the first line will be header
    # data_order is for order book and data_mess is for message book
    data_order_list.append(np.array(pd.read_csv(path_load+ticker_list[i]+name_order,header=-1),dtype="float64"))
    data_mess_list.append(np.array(pd.read_csv(path_load+ticker_list[i]+name_mess,header=-1),dtype="float64"))
    print("Time for importing the "+ticker_list[i]+" data is:",time.time()-t)
    print("The shape of the order data is: ",data_order_list[i].shape, " of message data is: ", data_mess_list[i].shape)
    # get the time index
    time_index_list.append(data_mess__list[i][:,0])


#print the sample of data
print("Check the original data:")

for i in range(len(ticker_list)):
    print()
    print("The first five sampe of "+ticker_list[i]+" is: ",data_order_list[i][:3])

    # -*- coding: utf-8 -*-

# # save the feature array
# ##get the original order,message and time index data, header =-1 means that did not
# ##read the first column as the name
#%%
# # use a loop to read data
# for ticker_ind in range(len(ticker_list)):
#     data_order=data_order_list[ticker_ind]
#     data_mess=data_mess_list[ticker_ind]
#     time_index=data_mess[:,0]
#     # obtain the reduced order message and time_index dataset, half an hour after the
#     # 9:30 and half an hour before 16:00
#     # data_reduced is used to install the data from 10 to 15:30, take half hour for auction
#     data_order_reduced=data_order[(time_index>= start_ind) & (time_index<= end_ind)]
#     data_mess_reduced=data_mess[(time_index>= start_ind) & (time_index<= end_ind)]
#     time_index_reduced=time_index[(time_index>= start_ind) & (time_index<= end_ind)]

#     test_lower=0
#     # test up is the up index of the original data to construct the test data
#     test_upper=len(data_order_reduced)
#     # data_test is the subset of data_reduced from the lower index to upper index
#     data_order_test=data_order_reduced[test_lower:test_upper,:]
#     data_mess_test=data_mess_reduced[test_lower:test_upper,:]
```

```
#     data_mess_test=data_mess_reduced[test_lower:test_upper,:]
#     t=time.time()
#     feature_array=get_features (data_order, data_mess,data_order_test,data_mess_test)
#     np.savetxt(path_save+ticker_list[ticker_ind]+'_feature_array.txt',feature_array,delimiter=' ')
#     print ("Time for building "+ticker_list[ticker_ind]+" is:",time.time()-t)


# load the feature
#%%
import time
t=time.time()
feature_array_list=[]
for ticker_ind in range(len(ticker_list)):
    feature_array_list.append(np.array(pd.read_csv(path_save+ticker_list[ticker_ind]+'_feature_array.txt',\
                                                   sep=' ',header=-1)))
print(time.time()-t)

# this function used to build the y
# ask_low as 1 bad high as -1 and no arbitrage as 0
# option=1 return ask low, option =2 return bid high, option =3 return no arbi, option =4 return total(ask_low=1,
# bid_high =-1 and no arbi =0)
#%%
def build_y(ask_low,bid_high,no_arbi,option):
    if (option==1):
        return ask_low
    elif option==2:
        return bid_high
    elif option==3:
        return no_arbi
    elif option==4:
        return ask_low-bid_high
    else:
        print("option should be 1,2,3,4")

## save y data
#%%
#time_ind=1
#option_ind=1
#for ticker_ind in range(len(ticker_list)):
#    response=build_y(ask_low_time_list[ticker_ind][time_ind],bid_high_time_list[ticker_ind][time_ind],\
#                             no_arbi_time_list[ticker_ind][time_ind],option=option_ind)
#    np.savetxt(path_save+ticker_list[ticker_ind]+'_response.txt',response)



## load y data
#%%
response_list=[]
for ticker_ind in range(len(ticker_list)):
    response_list.append((np.array(pd.read_csv(path_save+ticker_list[ticker_ind]+'_response.txt',header=-1))))


## print the shape of the response
```

```python
## note it is the total response
#%%
print("The shape of the total response is:\n")

for ticker_ind in range(len(ticker_list)):
    print(response_list[ticker_ind].shape)

# need to get the response from 10 to 15:30
# the shape of the response and the feature array should be equal
response_reduced_list=[]
for ticker_ind in range(len(ticker_list)):
    first_ind = np.where(time_index_list[ticker_ind]>=start_ind)[0][0]
    last_ind=np.where(time_index_list[ticker_ind]<=end_ind)[0][-1]
    response_reduced_list.append(response_list[ticker_ind][first_ind:last_ind+1])

print("The shape of the reduced response is:\n")

## print the shape of reduced response
## response reduced is used for testing and training the model
for ticker_ind in range(len(ticker_list)):
    print(response_reduced_list[ticker_ind].shape)
```

```
Time for importing the AAPL data is: 2.0496621131896973
The shape of the order data is:  (400391, 40)  of message data is:  (400391, 6)
Time for importing the AMZN data is: 1.4024591445922852
The shape of the order data is:  (269748, 40)  of message data is:  (269748, 6)
Time for importing the GOOG data is: 0.7670302391052246
The shape of the order data is:  (147916, 40)  of message data is:  (147916, 6)
Time for importing the INTC data is: 3.408174514770508
The shape of the order data is:  (624040, 40)  of message data is:  (624040, 6)
Time for importing the MSFT data is: 3.4592816829681396
The shape of the order data is:  (668765, 40)  of message data is:  (668765, 6)
Check the original data:

The first five sampe of AAPL is:  [[  5.85940000e+06   2.00000000e+02   5.85330000e+06   1.80000000e+01
    5.85980000e+06   2.00000000e+02   5.85300000e+06   1.50000000e+02
    5.86100000e+06   2.00000000e+02   5.85100000e+06   5.00000000e+00
    5.86890000e+06   3.00000000e+02   5.85010000e+06   8.90000000e+01
    5.86950000e+06   5.00000000e+01   5.84970000e+06   5.00000000e+00
    5.87000000e+06   1.00000000e+02   5.84930000e+06   3.00000000e+02
    5.87100000e+06   1.00000000e+01   5.84650000e+06   3.00000000e+02
    5.87390000e+06   1.00000000e+02   5.84530000e+06   3.00000000e+02
    5.87650000e+06   1.16000000e+03   5.84380000e+06   2.00000000e+02
    5.87900000e+06   5.00000000e+02   5.84270000e+06   3.00000000e+02]
 [  5.85940000e+06   2.00000000e+02   5.85330000e+06   1.80000000e+01
    5.85980000e+06   2.00000000e+02   5.85320000e+06   1.80000000e+01
    5.86100000e+06   2.00000000e+02   5.85300000e+06   1.50000000e+02
    5.86890000e+06   3.00000000e+02   5.85100000e+06   5.00000000e+00
    5.86950000e+06   5.00000000e+01   5.85010000e+06   8.90000000e+01
    5.87000000e+06   1.00000000e+02   5.84970000e+06   5.00000000e+00
    5.87100000e+06   1.00000000e+01   5.84930000e+06   3.00000000e+02
    5.87390000e+06   1.00000000e+02   5.84650000e+06   3.00000000e+02
    5.87650000e+06   1.16000000e+03   5.84530000e+06   3.00000000e+02
```

```
    5.87900000e+06   5.00000000e+02   5.84380000e+06   2.00000000e+02]
 [  5.85940000e+06   2.00000000e+02   5.85330000e+06   1.80000000e+01
    5.85980000e+06   2.00000000e+02   5.85320000e+06   1.80000000e+01
    5.86100000e+06   2.00000000e+02   5.85310000e+06   1.80000000e+01
    5.86890000e+06   3.00000000e+02   5.85300000e+06   1.50000000e+02
    5.86950000e+06   5.00000000e+01   5.85100000e+06   5.00000000e+00
    5.87000000e+06   1.00000000e+02   5.85010000e+06   8.90000000e+01
    5.87100000e+06   1.00000000e+01   5.84970000e+06   5.00000000e+00
    5.87390000e+06   1.00000000e+02   5.84930000e+06   3.00000000e+02
    5.87650000e+06   1.16000000e+03   5.84650000e+06   3.00000000e+02
    5.87900000e+06   5.00000000e+02   5.84530000e+06   3.00000000e+02]]

The first five sampe of AMZN is:  [[  2.23950000e+06   1.00000000e+02   2.23180000e+06   1.00000000e+02
    2.23990000e+06   1.00000000e+02   2.23070000e+06   2.00000000e+02
    2.24000000e+06   2.20000000e+02   2.23040000e+06   1.00000000e+02
    2.24250000e+06   1.00000000e+02   2.23000000e+06   1.00000000e+01
    2.24400000e+06   5.47000000e+02   2.22620000e+06   1.00000000e+02
    2.24540000e+06   1.00000000e+02   2.21300000e+06   4.00000000e+03
    2.24890000e+06   1.00000000e+02   2.20400000e+06   1.00000000e+02
    2.26770000e+06   1.00000000e+02   2.20250000e+06   5.00000000e+03
    2.29430000e+06   1.00000000e+02   2.20200000e+06   1.00000000e+02
    2.29800000e+06   1.00000000e+02   2.18970000e+06   1.00000000e+02]
 [  2.23950000e+06   1.00000000e+02   2.23810000e+06   2.10000000e+01
    2.23990000e+06   1.00000000e+02   2.23180000e+06   1.00000000e+02
    2.24000000e+06   2.20000000e+02   2.23070000e+06   2.00000000e+02
    2.24250000e+06   1.00000000e+02   2.23040000e+06   1.00000000e+02
    2.24400000e+06   5.47000000e+02   2.23000000e+06   1.00000000e+01
    2.24540000e+06   1.00000000e+02   2.22620000e+06   1.00000000e+02
    2.24890000e+06   1.00000000e+02   2.21300000e+06   4.00000000e+03
    2.26770000e+06   1.00000000e+02   2.20400000e+06   1.00000000e+02
    2.29430000e+06   1.00000000e+02   2.20250000e+06   5.00000000e+03
    2.29800000e+06   1.00000000e+02   2.20200000e+06   1.00000000e+02]
 [  2.23950000e+06   1.00000000e+02   2.23810000e+06   2.10000000e+01
    2.23960000e+06   2.00000000e+01   2.23180000e+06   1.00000000e+02
    2.23990000e+06   1.00000000e+02   2.23070000e+06   2.00000000e+02
    2.24000000e+06   2.20000000e+02   2.23040000e+06   1.00000000e+02
    2.24250000e+06   1.00000000e+02   2.23000000e+06   1.00000000e+01
    2.24400000e+06   5.47000000e+02   2.22620000e+06   1.00000000e+02
    2.24540000e+06   1.00000000e+02   2.21300000e+06   4.00000000e+03
    2.24890000e+06   1.00000000e+02   2.20400000e+06   1.00000000e+02
    2.26770000e+06   1.00000000e+02   2.20250000e+06   5.00000000e+03
    2.29430000e+06   1.00000000e+02   2.20200000e+06   1.00000000e+02]]

The first five sampe of GOOG is:  [[  5.80230000e+06   1.00000000e+02   5.79400000e+06   4.96000000e+02
    5.80430000e+06   1.00000000e+02   5.78700000e+06   4.00000000e+02
    5.80500000e+06   1.00000000e+02   5.78500000e+06   5.00000000e+02
    5.80630000e+06   1.00000000e+02   5.78000000e+06   5.00000000e+02
    5.80670000e+06   1.00000000e+02   5.77180000e+06   1.00000000e+02
    5.80960000e+06   5.00000000e+01   5.76940000e+06   1.00000000e+02
    5.80970000e+06   1.00000000e+02   5.76600000e+06   1.00000000e+02
    5.83500000e+06   1.00000000e+02   5.76260000e+06   1.00000000e+02
    5.88000000e+06   1.00000000e+02   5.73200000e+06   2.00000000e+01
    5.89260000e+06   1.00000000e+02   5.70000000e+06   1.00000000e+02]
```

```
 [  5.80230000e+06   1.00000000e+02   5.79400000e+06   1.96000000e+02
    5.80430000e+06   1.00000000e+02   5.78700000e+06   4.00000000e+02
    5.80500000e+06   1.00000000e+02   5.78500000e+06   5.00000000e+02
    5.80630000e+06   1.00000000e+02   5.78000000e+06   5.00000000e+02
    5.80670000e+06   1.00000000e+02   5.77180000e+06   1.00000000e+02
    5.80960000e+06   5.00000000e+01   5.76940000e+06   1.00000000e+02
    5.80970000e+06   1.00000000e+02   5.76600000e+06   1.00000000e+02
    5.83500000e+06   1.00000000e+02   5.76260000e+06   1.00000000e+02
    5.88000000e+06   1.00000000e+02   5.73200000e+06   2.00000000e+01
    5.89260000e+06   1.00000000e+02   5.70000000e+06   1.00000000e+02]
 [  5.80230000e+06   1.00000000e+02   5.79400000e+06   1.96000000e+02
    5.80430000e+06   1.00000000e+02   5.78700000e+06   4.00000000e+02
    5.80500000e+06   1.00000000e+02   5.78500000e+06   5.00000000e+02
    5.80630000e+06   1.00000000e+02   5.78000000e+06   5.00000000e+02
    5.80670000e+06   1.00000000e+02   5.77180000e+06   1.00000000e+02
    5.80960000e+06   5.00000000e+01   5.76940000e+06   1.00000000e+02
    5.80970000e+06   1.00000000e+02   5.76600000e+06   1.00000000e+02
    5.83500000e+06   1.00000000e+02   5.76260000e+06   1.00000000e+02
    5.88000000e+06   1.00000000e+02   5.73200000e+06   2.00000000e+01
    5.89260000e+06   1.00000000e+02   5.70000000e+06   1.00000000e+02]]

The first five sampe of INTC is:  [[  2.75200000e+05   6.60000000e+01   2.75100000e+05   4.00000000e+02
    2.75300000e+05   1.00000000e+03   2.75000000e+05   1.00000000e+02
    2.75400000e+05   3.73000000e+02   2.74900000e+05   2.00000000e+02
    2.75600000e+05   1.00000000e+02   2.74800000e+05   6.61000000e+02
    2.75700000e+05   1.00000000e+02   2.74700000e+05   3.00000000e+02
    2.75900000e+05   8.58900000e+03   2.74600000e+05   7.00000000e+02
    2.76000000e+05   9.59000000e+02   2.74500000e+05   9.00000000e+02
    2.76100000e+05   2.30000000e+03   2.74400000e+05   2.80000000e+03
    2.76200000e+05   2.70000000e+03   2.74300000e+05   3.30000000e+03
    2.76300000e+05   2.00000000e+03   2.74200000e+05   4.06300000e+03]
 [  2.75200000e+05   1.66000000e+02   2.75100000e+05   4.00000000e+02
    2.75300000e+05   1.00000000e+03   2.75000000e+05   1.00000000e+02
    2.75400000e+05   3.73000000e+02   2.74900000e+05   2.00000000e+02
    2.75600000e+05   1.00000000e+02   2.74800000e+05   6.61000000e+02
    2.75700000e+05   1.00000000e+02   2.74700000e+05   3.00000000e+02
    2.75900000e+05   8.58900000e+03   2.74600000e+05   7.00000000e+02
    2.76000000e+05   9.59000000e+02   2.74500000e+05   9.00000000e+02
    2.76100000e+05   2.30000000e+03   2.74400000e+05   2.80000000e+03
    2.76200000e+05   2.70000000e+03   2.74300000e+05   3.30000000e+03
    2.76300000e+05   2.00000000e+03   2.74200000e+05   4.06300000e+03]
 [  2.75200000e+05   1.66000000e+02   2.75100000e+05   4.00000000e+02
    2.75300000e+05   1.00000000e+03   2.75000000e+05   1.00000000e+02
    2.75400000e+05   3.73000000e+02   2.74900000e+05   2.00000000e+02
    2.75500000e+05   1.00000000e+02   2.74800000e+05   6.61000000e+02
    2.75600000e+05   1.00000000e+02   2.74700000e+05   3.00000000e+02
    2.75700000e+05   1.00000000e+02   2.74600000e+05   7.00000000e+02
    2.75900000e+05   8.58900000e+03   2.74500000e+05   9.00000000e+02
    2.76000000e+05   9.59000000e+02   2.74400000e+05   2.80000000e+03
    2.76100000e+05   2.30000000e+03   2.74300000e+05   3.30000000e+03
    2.76200000e+05   2.70000000e+03   2.74200000e+05   4.06300000e+03]]

The first five sampe of MSFT is:  [[  3.09900000e+05   3.78800000e+03   3.09500000e+05   3.00000000e+02
```

```
    3.10500000e+05   1.00000000e+02   3.09300000e+05   3.98600000e+03
    3.10600000e+05   1.00000000e+02   3.09200000e+05   1.00000000e+02
    3.10700000e+05   2.00000000e+02   3.09100000e+05   3.00000000e+02
    3.10800000e+05   2.00000000e+02   3.08900000e+05   1.00000000e+02
    3.10900000e+05   9.34800000e+03   3.08800000e+05   2.00000000e+02
    3.11000000e+05   1.80000000e+03   3.08700000e+05   2.00000000e+02
    3.11100000e+05   4.50000000e+03   3.08600000e+05   4.00000000e+02
    3.11300000e+05   1.00000000e+02   3.08500000e+05   4.00000000e+02
    3.11400000e+05   1.00000000e+02   3.08400000e+05   1.60000000e+03]
 [  3.09900000e+05   3.78800000e+03   3.09500000e+05   3.00000000e+02
    3.10500000e+05   2.00000000e+02   3.09300000e+05   3.98600000e+03
    3.10600000e+05   1.00000000e+02   3.09200000e+05   1.00000000e+02
    3.10700000e+05   2.00000000e+02   3.09100000e+05   3.00000000e+02
    3.10800000e+05   2.00000000e+02   3.08900000e+05   1.00000000e+02
    3.10900000e+05   9.34800000e+03   3.08800000e+05   2.00000000e+02
    3.11000000e+05   1.80000000e+03   3.08700000e+05   2.00000000e+02
    3.11100000e+05   4.50000000e+03   3.08600000e+05   4.00000000e+02
    3.11300000e+05   1.00000000e+02   3.08500000e+05   4.00000000e+02
    3.11400000e+05   1.00000000e+02   3.08400000e+05   1.60000000e+03]
 [  3.09900000e+05   3.78800000e+03   3.09500000e+05   3.00000000e+02
    3.10400000e+05   1.00000000e+02   3.09300000e+05   3.98600000e+03
    3.10500000e+05   2.00000000e+02   3.09200000e+05   1.00000000e+02
    3.10600000e+05   1.00000000e+02   3.09100000e+05   3.00000000e+02
    3.10700000e+05   2.00000000e+02   3.08900000e+05   1.00000000e+02
    3.10800000e+05   2.00000000e+02   3.08800000e+05   2.00000000e+02
    3.10900000e+05   9.34800000e+03   3.08700000e+05   2.00000000e+02
    3.11000000e+05   1.80000000e+03   3.08600000e+05   4.00000000e+02
    3.11100000e+05   4.50000000e+03   3.08500000e+05   4.00000000e+02
    3.11300000e+05   1.00000000e+02   3.08400000e+05   1.60000000e+03]]
80.32640743255615
The shape of the total response is:

(400236, 1)
(269571, 1)
(147766, 1)
(622641, 1)
(667701, 1)
The shape of the reduced response is:

(309538, 1)
(218710, 1)
(118877, 1)
(458160, 1)
(511299, 1)
```

2.train and test data split

In [3]:

```
# -*- coding: utf-8 -*-
# Random split
```

```
#%%-----------------------------------------------------------------------
import random
from sklearn.cross_validation import train_test_split

ticker_ind=1
size=100000

# combine the feature and response array to random sample
total_array=np.concatenate((feature_array_list[ticker_ind],response_reduced_list[ticker_ind]),axis=1)[:size,:]



print("total array shape:",total_array.shape)

#split the data to train and test data set
train_x, test_x, train_y, test_y =train_test_split(\
total_array[:,:134],total_array[:,134], test_size=0.1, random_state=42)

# the y data need to reshape to size (n,) not (n,1)
test_y=test_y.reshape(len(test_y),)
train_y=train_y.reshape(len(train_y),)

print("test_y shape:",test_y.shape)
print("train_y shape:",train_y.shape)
```

```
total array shape: (100000, 135)
test_y shape: (10000,)
train_y shape: (90000,)
```

In [77]:

```
np.random.choice(100,3,replace=False
                )
```

Out[77]:

```
array([35, 57, 46])
```

In [16]:

```
# random generate a given
def random_choice(num, key):
    temp=np.random.choice(num,size=key,replace=False)
    temp_sort=sorted(temp)
    for i in range(len(temp)):
        num[temp_sort[i]]=temp[i]

    return num
```

In [30]:

```
#time series split
#%%-----------------------------------------------------------------------------------------------
```

```
ticker_ind=1
size=100000
random_ratio=0.5
# combine the feature and response array to random sample
total_array=np.concatenate((feature_array_list[ticker_ind],response_reduced_list[ticker_ind]),axis=1)[:size,:]

total_array=total_array[random_choice(list(range(size)),int(size*random_ratio)),:]


train_num_index=int(len(total_array)*0.9)

print("total array shape:",total_array.shape)

#split the data to train and test data set
train_x=total_array[:train_num_index,:134]
test_x=total_array[train_num_index:,:134]
train_y=total_array[:train_num_index,134]
test_y=total_array[train_num_index:,134]


# the y data need to reshape to size (n,) not (n,1)
test_y=test_y.reshape(len(test_y),)
train_y=train_y.reshape(len(train_y),)
print("train_x shape:",train_x.shape)
print("test_x shape:",test_x.shape)
print("test_y shape:",test_y.shape)
print("train_y shape:",train_y.shape)
# scale data
#%%

# can use the processing.scale function to scale the data
from sklearn import preprocessing
# note that we need to transfer the data type to float
# remark: should use data_test=data_test.astype('float'),very important !!!!
# use scale for zero mean and one std
scaler = preprocessing.StandardScaler().fit(train_x)


train_x_scale=scaler.transform(train_x)
test_x_scale=scaler.transform(test_x)

print(np.mean(train_x_scale,0))
print(np.mean(test_x_scale,0))

# -*- coding: utf-8 -*-

# set the sample weights for the training model
sample_weights=[]
ratio=len(train_y)/sum(train_y==1)/10
for i in range(len(train_x)):
    if train_y[i]==0:
        sample_weights.append(1)
```

```
    else: sample_weights.append(ratio)
```

```
total array shape: (100000, 135)
train_x shape: (90000, 134)
test_x shape: (10000, 134)
test_y shape: (10000,)
train_y shape: (90000,)
[ -4.92e-14  -1.02e-14  -2.34e-14   6.61e-15   1.28e-14  -1.03e-14
   1.46e-14   9.94e-17  -3.27e-14   2.63e-15  -5.05e-15  -4.60e-15
  -2.69e-14  -8.17e-15  -1.48e-14   1.93e-15   4.11e-14  -6.04e-15
   2.70e-14   1.22e-15  -6.42e-15  -6.08e-15   3.10e-14  -6.34e-16
   3.26e-15  -1.15e-14  -1.09e-14  -9.13e-15  -2.37e-14  -6.47e-15
   3.40e-14   7.49e-15  -1.08e-15   2.01e-14   1.75e-14   3.22e-16
   2.86e-14   1.39e-14   1.19e-14   4.25e-15   5.71e-15   1.93e-14
   6.35e-15   9.73e-15   4.13e-15  -2.84e-14  -9.91e-15   5.47e-15
  -3.39e-15   3.19e-14  -2.40e-15   2.64e-15   2.90e-14  -1.84e-14
  -1.03e-15  -2.86e-14   4.12e-14   1.10e-14   8.49e-15  -1.21e-14
   1.42e-14  -1.12e-14  -4.83e-14   2.68e-15  -9.24e-16   1.17e-13
   5.32e-15  -5.99e-15   2.20e-14   1.31e-14   1.64e-14  -7.45e-14
  -7.04e-14  -2.44e-14   9.75e-14  -4.06e-14  -4.95e-14   3.28e-14
  -3.77e-14   1.91e-14  -8.49e-14   3.35e-15   1.78e-15  -2.86e-15
   1.16e-14   3.62e-15  -8.08e-15   1.10e-14  -4.35e-15  -6.69e-15
  -1.46e-15   3.88e-15  -1.71e-15  -4.36e-15  -1.00e-14  -8.94e-15
  -3.75e-15   7.71e-15  -1.31e-15  -5.85e-15  -7.51e-16   1.23e-15
   6.69e-16   3.25e-15   2.18e-15  -2.14e-15  -1.21e-15   3.25e-15
   1.60e-15   3.21e-15  -1.65e-16   2.54e-15   5.08e-15  -2.97e-15
   1.30e-15   4.76e-16   3.18e-16  -5.45e-16  -2.51e-15  -1.46e-15
  -3.12e-15  -3.95e-15   3.03e-15  -2.01e-15   1.84e-16  -1.15e-15
  -1.18e-14  -7.61e-15  -4.40e-16   3.60e-15   3.57e-16  -1.12e-15
  -3.21e-15   3.84e-16]
[-0.38 -0.17 -0.38 -0.04 -0.38 -0.19 -0.38 -0.03 -0.39 -0.12 -0.38 -0.03
 -0.39 -0.04 -0.37 -0.01 -0.39 -0.04 -0.37  0.   -0.39 -0.05 -0.36 -0.   -0.4
 -0.03 -0.36 -0.   -0.4   0.01 -0.35 -0.   -0.4  -0.02 -0.35 -0.02 -0.41
 -0.03 -0.35 -0.03  0.11  0.04 -0.05 -0.11 -0.16 -0.2  -0.24 -0.26 -0.28
 -0.3  -0.38 -0.38 -0.38 -0.38 -0.38 -0.38 -0.38 -0.38 -0.38 -0.38 -0.08
 -0.15 -0.1  -0.13 -0.13 -0.13 -0.14 -0.13 -0.13 -0.08 -0.11 -0.13 -0.15
 -0.15 -0.19 -0.19 -0.17 -0.17 -0.39 -0.36 -0.18 -0.06 -0.2   0.02  0.04
  0.03  0.02  0.04  0.04  0.04  0.03  0.04  0.03  0.06 -0.    0.01  0.01
  0.01  0.01 -0.    0.01  0.02  0.02  0.01 -0.01 -0.02  0.   -0.01 -0.
 -0.01  0.04 -0.05  0.04 -0.05 -0.    0.   -0.01 -0.    0.   -0.    0.01
 -0.01  0.02 -0.01 -0.09 -0.06 -0.06 -0.09 -0.07 -0.05 -0.02  0.01 -0.
 -0.03]
```

3.Model build

3.1 two classes

In [9]:

```
%matplotlib inline
```

logistic regression

In [10]:

```
train_x_scale.shape
```

Out[10]:

```
(90000, 134)
```

In [11]:

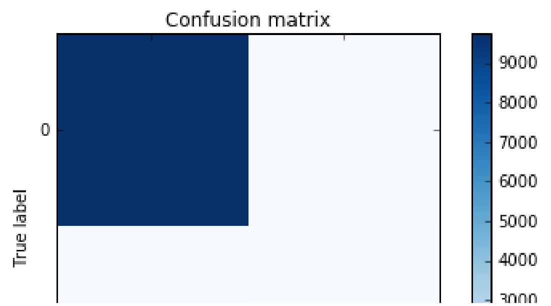
```
#----------------
# logistic l1
#-----------------


from sklearn import linear_model

# set the sample weights for the training model
sample_weights=[]
ratio=len(train_y)/sum(train_y==1)/10
for i in range(len(train_x)):
    if train_y[i]==0:
        sample_weights.append(1)
    else: sample_weights.append(ratio)

        # set the random state to make sure that each time get the same results

time_logistic=time.time()
clf = linear_model.LogisticRegression(C=1, penalty='l1', tol=1e-6,random_state= 987612345)
clf.fit(train_x_scale,train_y)
time_logistic=time.time()-time_logistic

print(time_logistic)

# test the training error
predict_y_logistic =np.array(clf.predict(train_x_scale))
print("train_accuracy is:",sum(predict_y_logistic==train_y)/len(train_y))

# test the score for the train data
from sklearn.metrics import (brier_score_loss, precision_score, recall_score,
                             f1_score)
precision= precision_score(predict_y_logistic,train_y)
recall = recall_score(predict_y_logistic,train_y)
f1=f1_score(predict_y_logistic,train_y)
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)

# define a function to prefict the result by threshold
```

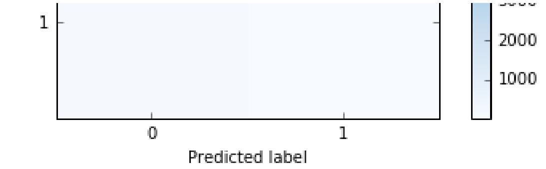
```python
# note: logistic model will return two probability
def predict_threshold(predict_proba, threshold):
    res=[]
    for i in range(len(predict_proba)):
        res.append(int(predict_proba[i][1]>threshold))
    return res

predict_y_test_proba =np.array(clf.predict_proba(test_x_scale))

predict_y_test=predict_threshold(predict_y_test_proba,0.5)

# test the score for the train data
from sklearn.metrics import (precision_score, recall_score,
                             f1_score)
print("accuracy is:",sum(predict_y_test==test_y)/len(test_y))
precision= precision_score(predict_y_test,test_y)
recall = recall_score(predict_y_test,test_y)
f1=f1_score(predict_y_test,test_y)
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)

%matplotlib inline
## draw chart for the cross table

from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(2)
    plt.xticks(tick_marks, [0,1])
    plt.yticks(tick_marks, [0,1])
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


# Compute confusion matrix
cm = confusion_matrix(test_y, predict_y_test)
np.set_printoptions(precision=2)
print('Confusion matrix, without normalization')
print(cm)
plt.figure()
plot_confusion_matrix(cm)
plt.show()

#-----------------
# logistic l2
#------------------
```

```
from sklearn import linear_model

# set the sample weights for the training model
sample_weights=[]
ratio=len(train_y)/sum(train_y==1)/10
for i in range(len(train_x)):
    if train_y[i]==0:
        sample_weights.append(1)
    else: sample_weights.append(ratio)

        # set the random state to make sure that each time get the same results

time_logistic=time.time()
clf = linear_model.LogisticRegression(C=1, penalty='l2', tol=1e-6,random_state= 987612345)
clf.fit(train_x_scale,train_y)
time_logistic=time.time()-time_logistic

print(time_logistic)

# test the training error
predict_y_logistic =np.array(clf.predict(train_x_scale))
print("train_accuracy is:",sum(predict_y_logistic==train_y)/len(train_y))

# test the score for the train data
from sklearn.metrics import (brier_score_loss, precision_score, recall_score,
                             f1_score)
precision= precision_score(predict_y_logistic,train_y)
recall = recall_score(predict_y_logistic,train_y)
f1=f1_score(predict_y_logistic,train_y)
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)

# define a function to prefict the result by threshold
# note: logistic model will return two probability
def predict_threshold(predict_proba, threshold):
    res=[]
    for i in range(len(predict_proba)):
        res.append(int(predict_proba[i][1]>threshold))
    return res

predict_y_test_proba =np.array(clf.predict_proba(test_x_scale))

predict_y_test=predict_threshold(predict_y_test_proba,0.5)

# test the score for the train data
from sklearn.metrics import (precision_score, recall_score,
                             f1_score)
print("accuracy is:",sum(predict_y_test==test_y)/len(test_y))
precision= precision_score(predict_y_test,test_y)
recall = recall_score(predict_y_test,test_y)
f1=f1_score(predict_y_test,test_y)
print("precision is: \t %s" % precision)
```

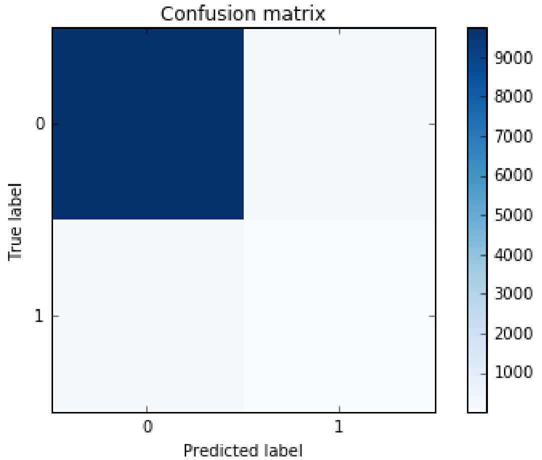
```
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)

%matplotlib inline
## draw chart for the cross table

from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(2)
    plt.xticks(tick_marks, [0,1])
    plt.yticks(tick_marks, [0,1])
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


# Compute confusion matrix
cm = confusion_matrix(test_y, predict_y_test)
np.set_printoptions(precision=2)
print('Confusion matrix, without normalization')
print(cm)
plt.figure()
plot_confusion_matrix(cm)
plt.show()
```

```
172.18734502792358
train_accuracy is: 0.984222222222
precision is:   0.489117043121
recall is:   0.871250914411
f1 score is:   0.626512361915
accuracy is: 0.9752
precision is:   0.122302158273
recall is:   0.118881118881
f1 score is:   0.120567375887
Confusion matrix, without normalization
[[9735  126]
 [ 122   17]]
```






```
12.392427921295166
train_accuracy is: 0.984144444444
precision is:   0.487474332649
recall is:   0.868960468521
f1 score is:   0.624572480926
accuracy is: 0.9761
precision is:   0.122302158273
recall is:   0.126865671642
f1 score is:   0.124542124542
Confusion matrix, without normalization
[[9744  117]
 [ 122   17]]
```



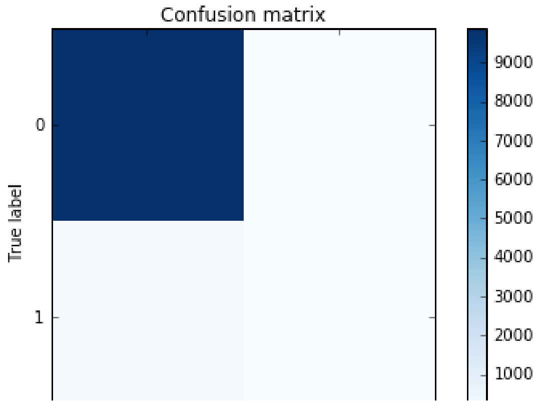
In [12]:

```
#--------------------
# SVM_poly_2
#---------------------

# set the sample weights for the training model
sample_weights=[]
ratio=len(train_y)/sum(train_y==1)/10
for i in range(len(train_x)):
    if train_y[i]==0:
        sample_weights.append(1)
    else: sample_weights.append(ratio)

import time
```

```
from sklearn import svm
# training

# change the depth of the tree to 6, number of estimators=100

t=time.time()
clf = svm.SVC(C=1.0,kernel='poly',degree=2,max_iter=5000,shrinking=True, tol=0.001, verbose=False)

clf.fit(train_x_scale,train_y)

print(time.time()-t)

#testing
# test the training error
predict_y =np.array(clf.predict(train_x_scale))
print("train_accuracy is:",sum(predict_y==train_y)/len(train_y))

# test the score for the train data
from sklearn.metrics import (brier_score_loss, precision_score, recall_score,
                             f1_score)
precision= precision_score(predict_y,train_y)
recall = recall_score(predict_y,train_y)
f1=f1_score(predict_y,train_y)
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)

# define a function to prefict the result by threshold
# note: logistic model will return two probability
def predict_threshold(predict_proba, threshold):
    res=[]
    for i in range(len(predict_proba)):
        res.append(int(predict_proba[i][1]>threshold))
    return res


predict_y_test=np.array(clf.predict(test_x_scale))


# test the score for the train data
from sklearn.metrics import (precision_score, recall_score,
                             f1_score)
print("accuracy is:",sum(predict_y_test==test_y)/len(test_y))
precision= precision_score(predict_y_test,test_y)
recall = recall_score(predict_y_test,test_y)
f1=f1_score(predict_y_test,test_y)
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)


#draw the crosstab chart
%matplotlib inline
```

```
%matplotlib inline
## draw chart for the cross table

from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(2)
    plt.xticks(tick_marks, [0,1])
    plt.yticks(tick_marks, [0,1])
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


# Compute confusion matrix
cm = confusion_matrix(test_y, predict_y_test)
np.set_printoptions(precision=2)
print('Confusion matrix, without normalization')
print(cm)
plt.figure()
plot_confusion_matrix(cm)
plt.show()
```

```
43.26794648170471
train_accuracy is: 0.988477777778
precision is:   0.606981519507
recall is:   0.9486521181
f1 score is:   0.740295517155
accuracy is: 0.9881
precision is:   0.143884892086
recall is:   1.0
f1 score is:   0.251572327044
Confusion matrix, without normalization
[[9861    0]
 [ 119   20]]
```



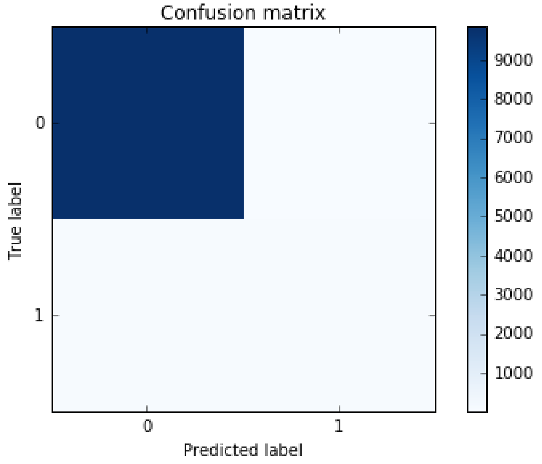
In [13]:

```
#---------------
# decision tree
#-----------------


# set the sample weights for the training model
sample_weights=[]
ratio=len(train_y)/sum(train_y==1)/10
for i in range(len(train_x)):
    if train_y[i]==0:
        sample_weights.append(1)
    else: sample_weights.append(ratio)


from sklearn import tree
# training

# change the depth of the tree to 6, number of estimators=100

t=time.time()
clf =  tree.DecisionTreeClassifier(max_depth=10,random_state= 987612345)
clf.fit(train_x_scale,train_y)

print(time.time()-t)

#testing
# test the training error
predict_y=np.array(clf.predict(train_x_scale))
print("train_accuracy is:",sum(predict_y==train_y)/len(train_y))

# test the score for the train data
from sklearn.metrics import (brier_score_loss, precision_score, recall_score,
                             f1_score)
precision= precision_score(predict_y,train_y)
recall = recall_score(predict_y,train_y)
f1=f1_score(predict_y,train_y)
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)

# define a function to prefict the result by threshold
# note: logistic model will return two probability
def predict_threshold(predict_proba, threshold):
    res=[]
    for i in range(len(predict_proba)):
        res.append(int(predict_proba[i][1]>threshold))
    return res
```

```
t=time.time()
predict_y_test_proba =np.array(clf.predict_proba(test_x_scale))
print("test time is:", time.time()-t)
predict_y_test=predict_threshold(predict_y_test_proba,0.5)


# test the score for the train data
from sklearn.metrics import (precision_score, recall_score,
                             f1_score)
print("accuracy is:",sum(predict_y_test==test_y)/len(test_y))
precision= precision_score(predict_y_test,test_y)
recall = recall_score(predict_y_test,test_y)
f1=f1_score(predict_y_test,test_y)
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)


#draw the crosstab chart
%matplotlib inline
## draw chart for the cross table

from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(2)
    plt.xticks(tick_marks, [0,1])
    plt.yticks(tick_marks, [0,1])
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


# Compute confusion matrix
cm = confusion_matrix(test_y, predict_y_test)
np.set_printoptions(precision=2)
print('Confusion matrix, without normalization')
print(cm)
plt.figure()
plot_confusion_matrix(cm)
plt.show()
```

```
2.9122314453125
train_accuracy is: 0.993455555556
precision is:   0.760164271047
recall is:   0.997306034483
f1 score is:   0.862735958984
test time is: 0.0032167434692382812
accuracy is: 0.9923
```

```
precision is:   0.58273381295
recall is:   0.81
f1 score is:   0.677824267782
Confusion matrix, without normalization
[[9842   19]
 [  58   81]]
```



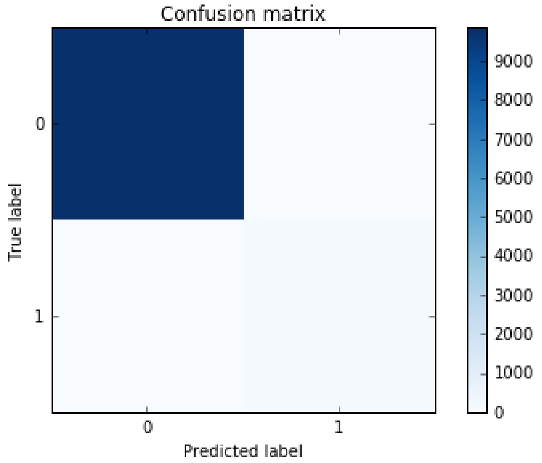
In [14]:

```
#----------------------------------------
# Adaboost
#----------------------------------------

# set the sample weights for the training model
sample_weights=[]
ratio=len(train_y)/sum(train_y==1)/10
for i in range(len(train_x)):
    if train_y[i]==0:
        sample_weights.append(1)
    else: sample_weights.append(ratio)


from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

# training

# change the depth of the tree to 6, number of estimators=100

time_ada=time.time()
clf =  AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=10),n_estimators=100,random_state= 987612345)
clf.fit(train_x_scale,train_y)

print(time.time()-time_ada)

#testing
```

```
# test the training error
predict_y=np.array(clf.predict(train_x_scale))
print("train_accuracy is:",sum(predict_y==train_y)/len(train_y))

# test the score for the train data
from sklearn.metrics import (brier_score_loss, precision_score, recall_score,
                             f1_score)
precision= precision_score(predict_y,train_y)
recall = recall_score(predict_y,train_y)
f1=f1_score(predict_y,train_y)
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)

# define a function to prefict the result by threshold
# note: logistic model will return two probability
def predict_threshold(predict_proba, threshold):
    res=[]
    for i in range(len(predict_proba)):
        res.append(int(predict_proba[i][1]>threshold))
    return res


predict_y_test_proba =np.array(clf.predict_proba(test_x_scale))

predict_y_test=predict_threshold(predict_y_test_proba,0.5)


# test the score for the train data
from sklearn.metrics import (precision_score, recall_score,
                             f1_score)
print("accuracy is:",sum(predict_y_test==test_y)/len(test_y))
precision= precision_score(predict_y_test,test_y)
recall = recall_score(predict_y_test,test_y)
f1=f1_score(predict_y_test,test_y)
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)


#draw the crosstab chart
%matplotlib inline
## draw chart for the cross table

from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(2)
    plt.xticks(tick_marks, [0,1])
    plt.yticks(tick_marks, [0,1])
```

```
    plt.yticks(tick_marks, [0,1])
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


# Compute confusion matrix
cm = confusion_matrix(test_y, predict_y_test)
np.set_printoptions(precision=2)
print('Confusion matrix, without normalization')
print(cm)
plt.figure()
plot_confusion_matrix(cm)
plt.show()
```

```
223.1659951210022
train_accuracy is: 0.999955555556
precision is:   0.998357289528
recall is:   1.0
f1 score is:   0.999177969585
accuracy is: 0.9995
precision is:   1.0
recall is:   0.965277777778
f1 score is:   0.982332155477
Confusion matrix, without normalization
[[9856    5]
 [   0  139]]
```



In [25]:

```
sum(predict_y)
```

Out[25]:

```
1741.0
```

In [31]:

```
# random forest

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

# training

# change the depth of the tree to 6, number of estimators=100

time_rf=time.time()
clf =  RandomForestClassifier(max_depth=20,n_estimators=100,random_state= 987612345)
clf.fit(train_x_scale,train_y)

print(time.time()-time_rf)

#testing
# test the training error
predict_y=np.array(clf.predict(train_x_scale))
print("train_accuracy is:",sum(predict_y==train_y)/len(train_y))

# test the score for the train data
from sklearn.metrics import (brier_score_loss, precision_score, recall_score,
                             f1_score)
precision= precision_score(predict_y,train_y)
recall = recall_score(predict_y,train_y)
f1=f1_score(predict_y,train_y)
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)

# define a function to prefict the result by threshold
# note: logistic model will return two probability
def predict_threshold(predict_proba, threshold):
    res=[]
    for i in range(len(predict_proba)):
        res.append(int(predict_proba[i][1]>threshold))
    return res

t=time.time()
predict_y_test_proba =np.array(clf.predict_proba(test_x_scale))
print("test time is:", time.time()-t)
predict_y_test=predict_threshold(predict_y_test_proba,0.5)


# test the score for the test data
from sklearn.metrics import (precision_score, recall_score,
                             f1_score)
print("test accuracy is:",sum(predict_y_test==test_y)/len(test_y))
precision= precision_score(predict_y_test,test_y)
recall = recall_score(predict_y_test,test_y)
f1=f1_score(predict_y_test,test_y)
print("precision is: \t %s" % precision)
```

```
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)
```

```
37.72304844856262
train_accuracy is: 0.997711111111
precision is:   0.900492610837
recall is:   0.997816593886
f1 score is:   0.946659761781
test time is: 0.10949921607971191
test accuracy is: 0.9965
precision is:   0.744
recall is:   0.96875
f1 score is:   0.841628959276
```

In [ ]:

```
## confusion matrix plot

from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(2)
    plt.xticks(tick_marks, [0,1])
    plt.yticks(tick_marks, [0,1])
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


# Compute confusion matrix
cm = confusion_matrix(test_y, predict_y_test)
np.set_printoptions(precision=2)
print('Confusion matrix, without normalization')
print(cm)
plt.figure()
plot_confusion_matrix(cm)
plt.show()
```

In [ ]:

```
## see the feature_importances

clf.feature_importances_
```

In [ ]:

```
test_x_scale[:100,-1]
```

3.2 Multi-class predict

3.2 Multi-class predict

In [78]:

```
## load the arbitrage time txt data

ask_low_time_list=[]
bid_high_time_list=[]
no_arbi_time_list=[]
time_list=[1,5,10,15,20]
import time
t=time.time()
for ticker_ind in range(5):
    ask_low_time_list.append([])
    bid_high_time_list.append([])
    no_arbi_time_list.append([])
    for time_ind in range(len(time_list)):
        ask_low_time_list[ticker_ind].append(
            np.array(pd.read_csv(path_save+ticker_list[ticker_ind]+'_ask_low_time_'+str(time_list[time_ind])+'.txt',header=-1)))
        bid_high_time_list[ticker_ind].append(
            np.array(pd.read_csv(path_save+ticker_list[ticker_ind]+'_bid_high_time_'+str(time_list[time_ind])+'.txt',header=-1)))
        no_arbi_time_list[ticker_ind].append(
            np.array(pd.read_csv(path_save+ticker_list[ticker_ind]+'_no_arbi_time_'+str(time_list[time_ind])+'.txt',header=-1)))

print(time.time()-t)
```

```
16.191282033920288
```

In [79]:

```
# Deal with the data
def build_y(ask_low,bid_high,no_arbi,option):
    if (option==1):
        return ask_low
    elif option==2:
        return bid_high
    elif option==3:
        return no_arbi
    elif option==4:
        return ask_low-bid_high
    else:
        print("option should be 1,2,3,4")

for ticker_ind in range(len(ticker_list)):
    response=build_y(ask_low_time_list[ticker_ind][1],bid_high_time_list[ticker_ind][1],\
                             no_arbi_time_list[ticker_ind][1],option=4)
    np.savetxt(path_save+ticker_list[ticker_ind]+'_multiresponse.txt',response)

response_list=[]
for ticker_ind in range(len(ticker_list)):
    response_list.append((np.array(pd.read_csv(path_save+ticker_list[ticker_ind]+'_multiresponse.txt',header=-1))))

    ## print the shape of the response
```

```python
## note it is the total response
print("The shape of the total response is:\n")

for ticker_ind in range(len(ticker_list)):
    print(response_list[ticker_ind].shape)

# need to get the response from 10 to 15:30
# the shape of the response and the feature array should be equal
response_reduced_list=[]
for ticker_ind in range(len(ticker_list)):
    first_ind = np.where(time_index_list[ticker_ind]>=start_ind)[0][0]
    last_ind=np.where(time_index_list[ticker_ind]<=end_ind)[0][-1]
    response_reduced_list.append(response_list[ticker_ind][first_ind:last_ind+1])

print("The shape of the reduced response is:\n")

## print the shape of reduced response
## response reduced is used for testing and training the model
for ticker_ind in range(len(ticker_list)):
    print(response_reduced_list[ticker_ind].shape)
    # random split data
```

```
The shape of the total response is:

(400236, 1)
(269571, 1)
(147766, 1)
(622641, 1)
(667701, 1)
The shape of the reduced response is:

(309538, 1)
(218710, 1)
(118877, 1)
(458160, 1)
(511299, 1)
```

In [ ]:

```python
# random split
#split the data to train and test data set
import random
from sklearn.cross_validation import train_test_split

ticker_ind=1
size=100000

# combine the feature and response array to random sample
total_array=np.concatenate((feature_array_list[ticker_ind],response_reduced_list[ticker_ind]),axis=1)[:size,:]


print("total shape:",total_array.shape)

train_x, test_x, train_y, test_y =train_test_split()
```

```
train_x, test_x, train_y, test_y =train_test_split(\
total_array[:,:134],total_array[:,134], test_size=0.1, random_state=42)

# the y data need to reshape to size (n,) not (n,1)
test_y=test_y.reshape(len(test_y),)
train_y=train_y.reshape(len(train_y),)

print("test shape:",test_y.shape)
print("train shape:",train_y.shape)
```

In [85]:

```
#time series split
#%%---------------------------------------------------------------------------------------

ticker_ind=1
size =100000
random_ratio=0.6

time_index=time_index_list[ticker_ind]
# combine the feature and response array to random sample
time_index_reduced=time_index[(time_index>=start_ind)&(time_index<=end_ind)]
total_array=np.concatenate((feature_array_list[ticker_ind],response_reduced_list[ticker_ind],
                            time_index_reduced.reshape(len(time_index_reduced),1)),axis=1)[:size,:]

total_array=total_array[random_choice(list(range(size)),int(size*random_ratio)),:]

total_array=total_array[np.random.randint(len(total_array),size=len(total_array)),:]

train_num_index=int(len(total_array)*0.9)

print("total array shape:",total_array.shape)

#split the data to train and test data set
train_x=total_array[:train_num_index,:134]
test_x=total_array[train_num_index:,:134]
train_y=total_array[:train_num_index,134]
test_y=total_array[train_num_index:,134]


# the y data need to reshape to size (n,) not (n,1)
test_y=test_y.reshape(len(test_y),)
train_y=train_y.reshape(len(train_y),)
print("train_x shape:",train_x.shape)
print("test_x shape:",test_x.shape)
print("test_y shape:",test_y.shape)
print("train_y shape:",train_y.shape)
# scale the data
# can use the processing.scale function to scale the data
from sklearn import preprocessing
# note that we need to transfer the data type to float
# remark: should use data_test=data_test.astype('float'),very important !!!!
# use scale for zero mean and one std
```

```
scaler = preprocessing.StandardScaler().fit(train_x)


train_x_scale=scaler.transform(train_x)
test_x_scale=scaler.transform(test_x)

print(np.mean(train_x_scale,0))
print(np.mean(test_x_scale,0))
```

```
total array shape: (100000, 136)
train_x shape: (90000, 134)
test_x shape: (10000, 134)
test_y shape: (10000,)
train_y shape: (90000,)
[  1.93e-14  -7.83e-16  -2.33e-14  -1.97e-16  -2.50e-14  -8.61e-16
  -5.91e-16   1.82e-16   1.36e-14  -1.26e-15   1.12e-14  -1.46e-15
   1.07e-14   4.74e-17   1.17e-14   2.89e-16  -3.90e-14   8.86e-16
  -2.96e-14  -1.39e-15   3.23e-15   4.24e-16   2.37e-14   1.12e-15
   2.10e-15  -3.35e-16   3.16e-14  -1.69e-15   2.46e-14   1.32e-15
  -4.33e-15  -6.33e-17  -1.60e-14  -1.46e-15   3.31e-15  -4.69e-17
   3.81e-14  -2.71e-20  -2.55e-14   3.56e-16  -1.64e-16   1.75e-16
  -8.13e-17   5.22e-16   8.61e-17   2.08e-16   1.63e-16   2.74e-16
  -4.26e-16  -3.75e-16  -2.48e-15   2.61e-14  -2.60e-14  -2.73e-14
   4.49e-15  -2.50e-14  -2.15e-14   9.84e-15  -6.19e-15  -3.32e-14
  -7.86e-16  -2.37e-15   1.12e-15   7.10e-15   1.40e-15  -6.09e-16
  -2.44e-16   9.95e-17  -4.22e-16  -3.44e-15  -7.05e-15  -3.82e-16
  -4.69e-16   3.64e-17   1.04e-15  -6.77e-16   1.00e-15   2.07e-15
  -3.56e-14  -3.02e-14  -4.33e-14  -1.07e-14   2.06e-16  -1.98e-18
   1.19e-15  -5.60e-16   6.97e-16   1.53e-16   9.89e-17  -1.45e-16
  -1.25e-16  -4.67e-16   6.29e-16  -2.61e-16  -5.40e-16   1.37e-16
  -3.90e-16   4.86e-16   1.33e-16  -1.46e-15  -9.86e-16  -8.68e-17
  -1.20e-15  -1.07e-15  -1.45e-15  -2.41e-17  -7.80e-16  -1.10e-16
  -1.43e-15  -1.41e-16   7.58e-16  -4.07e-16   1.81e-15   6.12e-17
   1.46e-15   4.95e-16  -6.96e-16  -8.65e-17   5.85e-16   1.72e-16
   6.27e-16  -1.93e-15  -2.10e-15  -4.47e-17   2.84e-16   1.27e-16
   6.02e-16  -1.83e-17  -4.64e-17  -3.57e-16  -8.07e-17  -1.85e-16
  -1.53e-15   2.28e-15]
[ 0.01 -0.    0.01  0.    0.01  0.01  0.01 -0.01  0.01 -0.01  0.01  0.
  0.01 -0.    0.01 -0.01  0.01 -0.01  0.01 -0.01  0.01 -0.    0.01  0.
  0.01  0.02  0.01 -0.01  0.01  0.    0.01 -0.    0.01 -0.01  0.01 -0.02
  0.01 -0.    0.01  0.    0.01  0.02  0.02  0.02  0.02  0.02  0.02  0.02
  0.02  0.02  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01  0.01
  0.01 -0.01 -0.    0.03  0.01  0.01  0.01  0.02  0.01  0.01  0.01  0.
 -0.01 -0.    0.   -0.    0.    0.01  0.01  0.01 -0.   -0.01  0.02  0.01
 -0.01 -0.01  0.    0.01  0.    0.    0.01  0.    0.   -0.    0.   -0.01
 -0.01 -0.01 -0.   -0.01  0.   -0.   -0.    0.01  0.01  0.   -0.01  0.01
 -0.   -0.01  0.02 -0.01 -0.01 -0.01 -0.01  0.01  0.01 -0.   -0.02 -0.01
 -0.   -0.    0.01  0.03 -0.01 -0.01 -0.01 -0.01 -0.01 -0.01 -0.02 -0.02
 -0.02 -0.   ]
```

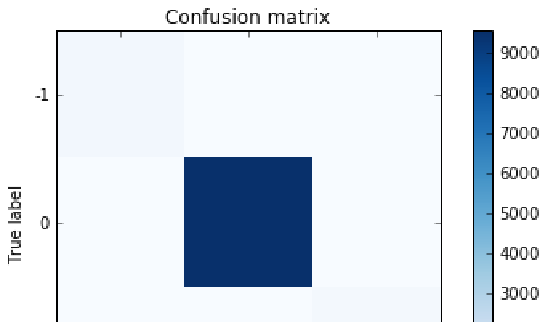
one vs one

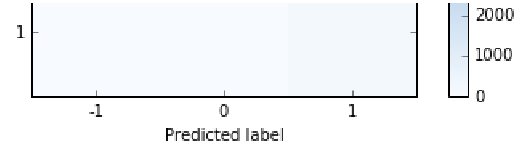
In [86]:

```
# only run for random forest method
# one vs one case
# random forest
from sklearn.multiclass import OneVsRestClassifier,OneVsOneClassifier
from sklearn.ensemble import RandomForestClassifier


## sample weights
#sample_weights=[]
#ratio=len(train_y)/sum(train_y==1)/10
#for i in range(len(train_x)):
#    if train_y[i]==0:
#        sample_weights.append(1)
#    else: sample_weights.append(ratio)


# training

# change the depth of the tree to 6, number of estimators=100

t=time.time()
clf =  OneVsOneClassifier(RandomForestClassifier(max_depth=20,n_estimators=100,random_state= 987612345))
clf.fit(train_x_scale,train_y)

print(time.time()-t)

predict_y_test=np.array(clf.predict(train_x_scale))

print("train accuracy is:",sum(predict_y_test==train_y)/len(train_y))

# define a function to prefict the result by threshold
# note: logistic model will return two probability
def predict_threshold(predict_proba, threshold):
    res=[]
    for i in range(len(predict_proba)):
        res.append(int(predict_proba[i][1]>threshold))
    return res

t=time.time()
predict_y_test=np.array(clf.predict(test_x_scale))
print("test time is :",time.time()-t)

print("test accuracy is:",sum(predict_y_test==test_y)/len(test_y))

# # test the score for the train data
# from sklearn.metrics import (precision_score, recall_score,
#                              f1_score)
# print("test accuracy is:",sum(predict_y_test==test_y)/len(test_y))
# precision= precision_score(predict_y_test,test_y)
# recall = recall_score(predict_y_test,test_y)
# f1=f1_score(predict_y_test,test_y)
```

```
# f1=f1_score(predict_y_test,test_y)
# print("precision is: \t %s" % precision)
# print("recall is: \t %s" % recall)
# print("f1 score is: \t %s" %f1)


# #draw the crosstab chart
# %matplotlib inline
# ## draw chart for the cross table
from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(3)
    plt.xticks(tick_marks, [-1,0,1])
    plt.yticks(tick_marks, [-1,0,1])
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

%matplotlib inline
# Compute confusion matrix
cm = confusion_matrix(test_y, predict_y_test)
np.set_printoptions(precision=2)
print('Confusion matrix, without normalization')
print(cm)
plt.figure()
plot_confusion_matrix(cm)
plt.savefig("one_vs_one.png")
plt.show()
```

```
68.89510798454285
train accuracy is: 0.996855555556
test time is : 0.6284787654876709
test accuracy is: 0.9958
Confusion matrix, without normalization
[[ 252   13    0]
 [   1 9533    3]
 [   0   25  173]]
```





In [ ]:

```
# one vs one case
# adaboosting
from sklearn.multiclass import OneVsRestClassifier,OneVsOneClassifier
from sklearn.ensemble import AdaBoostClassifier


## sample weights
#sample_weights=[]
#ratio=len(train_y)/sum(train_y==1)/10
#for i in range(len(train_x)):
#    if train_y[i]==0:
#        sample_weights.append(1)
#    else: sample_weights.append(ratio)


# training

# change the depth of the tree to 6, number of estimators=100

t=time.time()
clf = OneVsOneClassifier( AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=10),n_estimators=100,random_state= 987612345))
clf.fit(train_x_scale,train_y)

print(time.time()-t)

predict_y_test=np.array(clf.predict(train_x_scale))

print("train accuracy is:",sum(predict_y_test==train_y)/len(train_y))

# define a function to prefict the result by threshold
# note: logistic model will return two probability
def predict_threshold(predict_proba, threshold):
    res=[]
    for i in range(len(predict_proba)):
        res.append(int(predict_proba[i][1]>threshold))
    return res

t=time.time()
predict_y_test=np.array(clf.predict(test_x_scale))
print("test time is :",time.time()-t)

print("test accuracy is:",sum(predict_y_test==test_y)/len(test_y))

# # test the score for the train data
```

```
# from sklearn.metrics import (precision_score, recall_score,
#                               f1_score)
# print("test accuracy is:",sum(predict_y_test==test_y)/len(test_y))
# precision= precision_score(predict_y_test,test_y)
# recall = recall_score(predict_y_test,test_y)
# f1=f1_score(predict_y_test,test_y)
# print("precision is: \t %s" % precision)
# print("recall is: \t %s" % recall)
# print("f1 score is: \t %s" %f1)


# #draw the crosstab chart
# %matplotlib inline
# ## draw chart for the cross table
from sklearn.metrics import confusion_matrix
%matplotlib inline
def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(3)
    plt.xticks(tick_marks, [-1,0,1])
    plt.yticks(tick_marks, [-1,0,1])
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


# Compute confusion matrix
cm = confusion_matrix(test_y, predict_y_test)
np.set_printoptions(precision=2)
print('Confusion matrix, without normalization')
print(cm)
plt.figure()
plot_confusion_matrix(cm)
plt.savefig("one_vs_one.png")
plt.show()
```

In [ ]:

```
#----------------
# one vs one case
# svm
#-------------------
from sklearn.multiclass import OneVsRestClassifier,OneVsOneClassifier


## sample weights
#sample_weights=[]
#ratio=len(train_y)/sum(train_y==1)/10
#for i in range(len(train_x)):
#    if train_y[i]==0:
#        sample_weights.append(1)
```

```
#    else: sample_weights.append(ratio)


# training

# change the depth of the tree to 6, number of estimators=100

t=time.time()
clf =  OneVsOneClassifier(svm.SVC(C=1.0,kernel='poly',degree=2,max_iter=5000,shrinking=True, tol=0.001, verbose=False)
)
clf.fit(train_x_scale,train_y)

print(time.time()-t)

predict_y_test=np.array(clf.predict(train_x_scale))

print("train accuracy is:",sum(predict_y_test==train_y)/len(train_y))

# define a function to prefict the result by threshold
# note: logistic model will return two probability
def predict_threshold(predict_proba, threshold):
    res=[]
    for i in range(len(predict_proba)):
        res.append(int(predict_proba[i][1]>threshold))
    return res

t=time.time()
predict_y_test=np.array(clf.predict(test_x_scale))
print("test time is :",time.time()-t)

print("test accuracy is:",sum(predict_y_test==test_y)/len(test_y))

# # test the score for the train data
# from sklearn.metrics import (precision_score, recall_score,
#                              f1_score)
# print("test accuracy is:",sum(predict_y_test==test_y)/len(test_y))
# precision= precision_score(predict_y_test,test_y)
# recall = recall_score(predict_y_test,test_y)
# f1=f1_score(predict_y_test,test_y)
# print("precision is: \t %s" % precision)
# print("recall is: \t %s" % recall)
# print("f1 score is: \t %s" %f1)


# #draw the crosstab chart
# %matplotlib inline
# ## draw chart for the cross table
from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
```

```
    plt.colorbar()
    tick_marks = np.arange(3)
    plt.xticks(tick_marks, [-1,0,1])
    plt.yticks(tick_marks, [-1,0,1])
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


# Compute confusion matrix
cm = confusion_matrix(test_y, predict_y_test)
np.set_printoptions(precision=2)
print('Confusion matrix, without normalization')
print(cm)
plt.figure()
plot_confusion_matrix(cm)
plt.savefig("one_vs_one.png")
plt.show()
```

One vs rest

In [ ]:

```
# only run for random forest method
# one vs rest case
from sklearn.multiclass import OneVsRestClassifier,OneVsOneClassifier





# change the depth of the tree to 6, number of estimators=100

t=time.time()
clf =  OneVsRestClassifier(RandomForestClassifier(max_depth=20,n_estimators=100,random_state= 987612345))
clf.fit(train_x_scale,train_y)

print(time.time()-t)

predict_y_test=np.array(clf.predict(train_x_scale))

print("train accuracy is:",sum(predict_y_test==train_y)/len(train_y))

# define a function to pbrefict the result by threshold
# note: logistic model will return two probability
def predict_threshold(predict_proba, threshold):
    res=[]
    for i in range(len(predict_proba)):
        res.append(int(predict_proba[i][1]>threshold))
    return res

t=time.time()
```

```
predict_y_test=np.array(clf.predict(test_x_scale))
print("test time is :",time.time()-t)
print("test accuracy is:",sum(predict_y_test==test_y)/len(test_y))

# # test the score for the train data
# from sklearn.metrics import (precision_score, recall_score,
#                              f1_score)
# print("test accuracy is:",sum(predict_y_test==test_y)/len(test_y))
# precision= precision_score(predict_y_test,test_y)
# recall = recall_score(predict_y_test,test_y)
# f1=f1_score(predict_y_test,test_y)
# print("precision is: \t %s" % precision)
# print("recall is: \t %s" % recall)
# print("f1 score is: \t %s" %f1)


# #draw the crosstab chart
# %matplotlib inline
# ## draw chart for the cross table
from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(3)
    plt.xticks(tick_marks, [-1,0,1])
    plt.yticks(tick_marks, [-1,0,1])
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


# Compute confusion matrix
cm = confusion_matrix(test_y, predict_y_test)
np.set_printoptions(precision=2)
print('Confusion matrix, without normalization')
print(cm)
plt.figure()
plot_confusion_matrix(cm)
plt.savefig("one_vs_rest.png")
plt.show()
```

In [ ]:

```
# one vs one case
# adaboosting
from sklearn.multiclass import OneVsRestClassifier,OneVsOneClassifier
from sklearn.ensemble import AdaBoostClassifier


## sample weights
#sample_weights=[]
```

```
#sample_weights=[]
#ratio=len(train_y)/sum(train_y==1)/10
#for i in range(len(train_x)):
#    if train_y[i]==0:
#        sample_weights.append(1)
#    else: sample_weights.append(ratio)


# training

# change the depth of the tree to 6, number of estimators=100

t=time.time()
clf = OneVsRestClassifier(AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=10),n_estimators=100,random_state= 987612345))
clf.fit(train_x_scale,train_y)

print(time.time()-t)

predict_y_test=np.array(clf.predict(train_x_scale))

print("train accuracy is:",sum(predict_y_test==train_y)/len(train_y))

# define a function to prefict the result by threshold
# note: logistic model will return two probability
def predict_threshold(predict_proba, threshold):
    res=[]
    for i in range(len(predict_proba)):
        res.append(int(predict_proba[i][1]>threshold))
    return res

t=time.time()
predict_y_test=np.array(clf.predict(test_x_scale))
print("test time is :",time.time()-t)

print("test accuracy is:",sum(predict_y_test==test_y)/len(test_y))

# # test the score for the train data
# from sklearn.metrics import (precision_score, recall_score,
#                              f1_score)
# print("test accuracy is:",sum(predict_y_test==test_y)/len(test_y))
# precision= precision_score(predict_y_test,test_y)
# recall = recall_score(predict_y_test,test_y)
# f1=f1_score(predict_y_test,test_y)
# print("precision is: \t %s" % precision)
# print("recall is: \t %s" % recall)
# print("f1 score is: \t %s" %f1)


# #draw the crosstab chart
# %matplotlib inline
# ## draw chart for the cross table
from sklearn.metrics import confusion_matrix
%matplotlib inline
```

```
def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(3)
    plt.xticks(tick_marks, [-1,0,1])
    plt.yticks(tick_marks, [-1,0,1])
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


# Compute confusion matrix
cm = confusion_matrix(test_y, predict_y_test)
np.set_printoptions(precision=2)
print('Confusion matrix, without normalization')
print(cm)
plt.figure()
plot_confusion_matrix(cm)
plt.savefig("one_vs_one.png")
plt.show()
```

4.P&L calculation

In [87]:

```
def get_index(index, value):
    i=0
    while index[i] <value:
        i=i+1
    return i
```

In [90]:

```
## for AMZN
ticker_ind =1
train_ratio=0.9
data_mess=data_mess_list[ticker_ind]
data_order=data_order_list[ticker_ind]

time_index=data_mess[:,0]
data_order_reduced=data_order[(time_index>= start_ind) & (time_index<= end_ind)]
time_index_reduced=time_index[(time_index>= start_ind) & (time_index<= end_ind)]
total_array_old=np.concatenate((feature_array_list[ticker_ind],response_reduced_list[ticker_ind],
                                time_index_reduced.reshape(len(time_index_reduced),1)),axis=1)
```

In [91]:

```
test_y.shape
```

Out[91]:

(10000,)

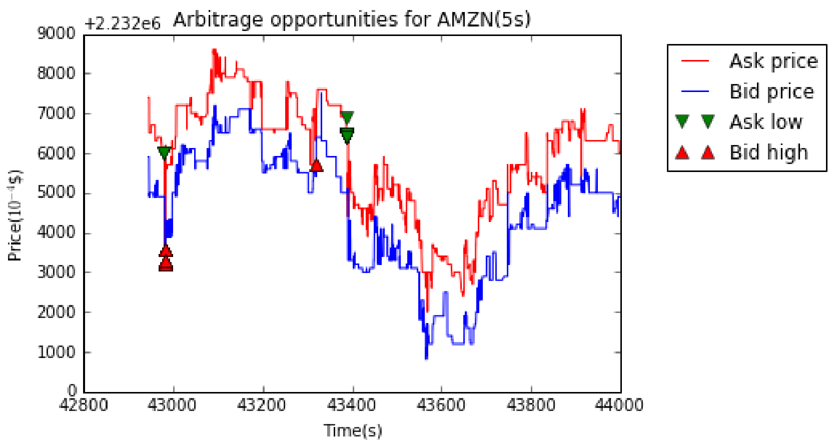
In [92]:

```
data_order_test=data_order_reduced[int(size*train_ratio):size,:]
time_index_test=time_index_reduced[int(size*train_ratio):size]

test_y_unrandom=total_array_old[int(size*train_ratio):size,134]
print(data_order_test.shape)
print(time_index_test.shape)
```

(10000, 40)
(10000,)

In [94]:

```
import matplotlib.pyplot as plt

plt.plot(time_index_test[:10000],data_order_test[:10000,0],"r-",label="Ask price")
plt.plot(time_index_test[:10000],data_order_test[:10000,2],"b-",label="Bid price")

x_ask_low_choose=time_index_test[test_y_unrandom==1]
y_ask_low_choose=data_order_test[test_y_unrandom==1,0]
x_bid_high_choose=time_index_test[test_y_unrandom==-1]
y_bid_high_choose=data_order_test[test_y_unrandom==-1,2]

plt.plot(x_ask_low_choose[:30],y_ask_low_choose[:30],"gv",markersize=8,label="Ask low")
plt.plot(x_bid_high_choose[:30],y_bid_high_choose[:30],"r^",markersize=8,label="Bid high")
plt.xlabel("Time(s)")
plt.ylabel("Price($10^{-4}$\$)")
plt.legend(bbox_to_anchor=[1.4, 1])
plt.title("Arbitrage opportunities for "+ticker_list[ticker_ind]+"(5s)")
plt.savefig("arbitrage_plot.png")
plt.show()
```



In [95]:

In [95]:

```python
time_index_test=total_array[:,135][int(size*train_ratio):size]
# find the arbitrage occuring index
arbi_index=list(np.where(predict_y_test!=0)[0])
# find the index that 5 seconds later
arbi_future_index=[]
for i in arbi_index:
    arbi_future_index.append(get_index(time_index_reduced,time_index_test[i]+5))
```

In [ ]:

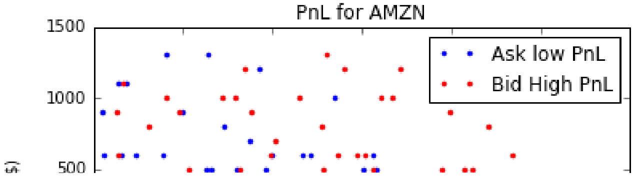
```python
arbi_future_index
```

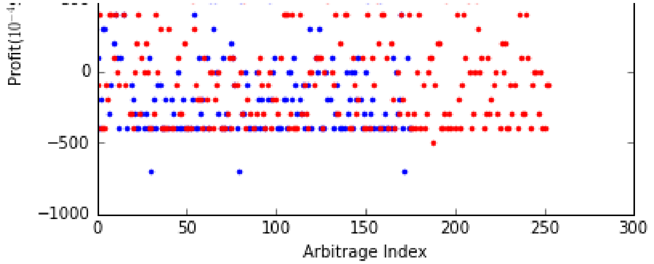
In [115]:

```python
total_array_test=total_array[int(size*train_ratio):size,:]
future_price=[]
current_price=[]
pnl=[]
cost=500
for i in range(len(arbi_index)):
    #ask low
    if predict_y_test[arbi_index[i]]==1 :
        future_price=data_order_reduced[arbi_future_index[i],0]
        current_price=total_array_test[arbi_index[i],2]
        pnl.append(current_price-future_price-cost)
    # bid high
    else:
        future_price=data_order_reduced[arbi_future_index[i],2]
        current_price=total_array_test[arbi_index[i],0]
        pnl.append(future_price-current_price-cost)
```

In [116]:

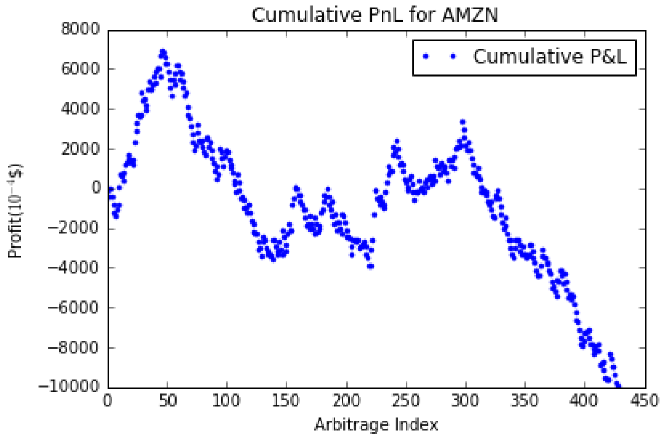
```python
pnl=np.array(pnl)
predict_arbi=predict_y_test[predict_y_test!=0]
plt.plot(pnl[predict_arbi==1],"b.",label="Ask low PnL")
plt.plot(pnl[predict_arbi==-1],"r.",label="Bid High PnL")

plt.xlabel("Arbitrage Index")
plt.ylabel("Profit($10^{-4}$\$)")
plt.title("PnL for "+ticker_list[ticker_ind])
plt.legend()
plt.savefig(ticker_list[ticker_ind]+"_pnl.png")
plt.show()
```





In [117]:

```
cum_pnl=np.cumsum(pnl)
plt.plot(cum_pnl,"b.",label="Cumulative P&L")
plt.xlabel("Arbitrage Index")
plt.ylabel("Profit($10^{-4}$\$)")
plt.title("Cumulative PnL for "+ticker_list[ticker_ind])
plt.legend()
plt.savefig(ticker_list[ticker_ind]+"_cum_pnl.png")
plt.show()
```



loop for all stock to plot the pnl

In [ ]:

```
#time series split
#%%-----------------------------------------------------------------------------------------

size =100000
for ticker_ind in range(2,5):
    # combine the feature and response array to random sample
    data_order=data_order_list[ticker_ind]
    data_mess=data_mess_list[ticker_ind]
```

```
time_index=data_mess[:,0]
data_order_reduced=data_order[(time_index>= start_ind) & (time_index<= end_ind)]
time_index_reduced=time_index[(time_index>= start_ind) & (time_index<= end_ind)]
total_array_old=np.concatenate((feature_array_list[ticker_ind],response_reduced_list[ticker_ind],
                                time_index_reduced.reshape(len(time_index_reduced),1)),axis=1)

total_array=np.concatenate((feature_array_list[ticker_ind],response_reduced_list[ticker_ind],
                            time_index_reduced.reshape(len(time_index_reduced),1)),axis=1)[:size,:]
total_array=total_array[np.random.randint(len(total_array),size=len(total_array)),:]

train_num_index=int(len(total_array)*0.9)

print("total array shape:",total_array.shape)

#split the data to train and test data set
train_x=total_array[:train_num_index,:134]
test_x=total_array[train_num_index:,:134]
train_y=total_array[:train_num_index,134]
test_y=total_array[train_num_index:,134]


# the y data need to reshape to size (n,) not (n,1)
test_y=test_y.reshape(len(test_y),)
train_y=train_y.reshape(len(train_y),)
print("train_x shape:",train_x.shape)
print("test_x shape:",test_x.shape)
print("test_y shape:",test_y.shape)
print("train_y shape:",train_y.shape)


# scale the data
# can use the processing.scale function to scale the data
from sklearn import preprocessing
# note that we need to transfer the data type to float
# remark: should use data_test=data_test.astype('float'),very important !!!!
# use scale for zero mean and one std
scaler = preprocessing.StandardScaler().fit(train_x)


train_x_scale=scaler.transform(train_x)
test_x_scale=scaler.transform(test_x)

print(np.mean(train_x_scale,0))
print(np.mean(test_x_scale,0))

from sklearn.multiclass import OneVsRestClassifier,OneVsOneClassifier
# change the depth of the tree to 6, number of estimators=100

t=time.time()
clf =  OneVsRestClassifier(RandomForestClassifier(max_depth=20,n_estimators=100,random_state= 987612345))
clf.fit(train_x_scale,train_y)

print(time.time()-t)
```

```
predict_y_test=np.array(clf.predict(train_x_scale))

print("train accuracy is:",sum(predict_y_test==train_y)/len(train_y))

# define a function to prefict the result by threshold
# note: logistic model will return two probability
def predict_threshold(predict_proba, threshold):
    res=[]
    for i in range(len(predict_proba)):
        res.append(int(predict_proba[i][1]>threshold))
    return res

t=time.time()
predict_y_test=np.array(clf.predict(test_x_scale))
print("test time is :",time.time()-t)
print("test accuracy is:",sum(predict_y_test==test_y)/len(test_y))

# # test the score for the train data
# from sklearn.metrics import (precision_score, recall_score,
#                              f1_score)
# print("test accuracy is:",sum(predict_y_test==test_y)/len(test_y))
# precision= precision_score(predict_y_test,test_y)
# recall = recall_score(predict_y_test,test_y)
# f1=f1_score(predict_y_test,test_y)
# print("precision is: \t %s" % precision)
# print("recall is: \t %s" % recall)
# print("f1 score is: \t %s" %f1)


# #draw the crosstab chart
# %matplotlib inline
# ## draw chart for the cross table
from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(3)
    plt.xticks(tick_marks, [-1,0,1])
    plt.yticks(tick_marks, [-1,0,1])
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


# Compute confusion matrix
cm = confusion_matrix(test_y, predict_y_test)
np.set_printoptions(precision=2)
print('Confusion matrix, without normalization')
print(cm)
plt.figure()
```

```
plt.figure()
plot_confusion_matrix(cm)
plt.savefig("one_vs_rest.png")
plt.show()

def get_index(index, value):
    i=0
    while index[i] <value:
        i=i+1
    return i


train_ratio=0.9
time_index=data_mess[:,0]
data_order_reduced=data_order[(time_index>= start_ind) & (time_index<= end_ind)]
time_index_reduced=time_index[(time_index>= start_ind) & (time_index<= end_ind)]
total_array_old=np.concatenate((feature_array_list[ticker_ind],response_reduced_list[ticker_ind],
                                time_index_reduced.reshape(len(time_index_reduced),1)),axis=1)
data_order=data_order_list[ticker_ind]
data_mess=data_mess_list[ticker_ind]

time_index_test=total_array[:,135][int(size*train_ratio):size]
# find the arbitrage occuring index
arbi_index=list(np.where(predict_y_test!=0)[0])
# find the index that 5 seconds later
arbi_future_index=[]
for i in arbi_index:
    arbi_future_index.append(get_index(time_index_reduced,time_index_test[i]+5))

total_array_test=total_array[int(size*train_ratio):size,:]
future_price=[]
current_price=[]
pnl=[]
for i in range(len(arbi_index)):
    #ask low
    if predict_y_test[arbi_index[i]]==1 :
        future_price=data_order_reduced[arbi_future_index[i],0]
        current_price=total_array_test[arbi_index[i],2]
        pnl.append(current_price-future_price)
    # bid high
    else:
        future_price=data_order_reduced[arbi_future_index[i],2]
        current_price=total_array_test[arbi_index[i],0]
        pnl.append(future_price-current_price)

pnl=np.array(pnl)
predict_arbi=predict_y_test[predict_y_test!=0]
plt.plot(pnl[predict_arbi==1],"b.",label="Ask low PnL")
plt.plot(pnl[predict_arbi==-1],"r.",label="Bid High PnL")

plt.xlabel("Arbitrage Index")
plt.ylabel("Profit($10^{-4}$\$)")
plt.title("PnL for "+ticker_list[ticker_ind])
plt.legend()
```

```
    plt.legend()
    plt.savefig(ticker_list[ticker_ind]+"_pnl.png")
    plt.show()

    cum_pnl=np.cumsum(pnl)
    plt.plot(cum_pnl,"b.",label="Cumulative P&L")
    plt.xlabel("Arbitrage Index")
    plt.ylabel("Profit($10^{-4}$\$)")
    plt.title("Cumulative PnL for "+ticker_list[ticker_ind])
    plt.legend()
    plt.savefig(ticker_list[ticker_ind]+"_cum_pnl.png")
    plt.show()
```

Plot the order book type

use the data_mess data set to plot the chart of the order book type

Plot the volume, visible, hidden, depth and snap shot

In [ ]:

```
# fun for total

#---------------------------------------------------------------------------
#%%  set the parameters
#Stock name
#---------------------------------------------------------------------------
ticker ="INTC"


#---------------------------------------------------------------------------
# Levels
#---------------------------------------------------------------------------
lvl= 10

#---------------------------------------------------------------------------
# File names
#---------------------------------------------------------------------------
path='/media/jianwang/Study1/Research/order_book/'
path_save='/media/jianwang/Study1/Research/order_book/'
path_save='/media/jianwang/Study1/Research/order_book/'
name_book    = 'AMZN_2012-06-21_34200000_57600000_orderbook_10.csv'
name_mess    = 'AMZN_2012-06-21_34200000_57600000_message_10.csv'

#----------------------------------------------------------------------------
# Date of files
#----------------------------------------------------------------------------
demo_date    = [2012,6,21]    #year, month, day

#-----------------------------------------------------------------------------
```

```
# Load Messsage File
#---------------------------------------------------------------------------
#  Load data
t=time.time()
mess = np.array(pd.read_csv(path+name_mess))
print("The time for reading the CSV file",time.time()-t)
#
#
#% Message file information:
#% ----------------------------------------------------------
#%
#%   - Dimension:    (NumberEvents x 6)
#%
#%   - Structure:    Each row:
#%                   Time stamp (sec after midnight with decimal
#%                   precision of at least milliseconds and
#%                   up to nanoseconds depending on the period),
#%                   Event type, Order ID, Size (# of shares),
#%                   Price, Direction
#%
#%                   Event types:
#%                       - '1'   Submission new limit order
#%                       - '2'   Cancellation (partial)
#%                       - '3'   Deletion (total order)
#%                       - '4'   Execution of a visible limit order
#%                       - '5'   Execution of a hidden limit order
#%                               liquidity
#%                       - '7'   Trading Halt (Detailed
#%                               information below)
#%
#%                   Direction:
#%                       - '-1'  Sell limit order
#%                       - '-2'  Buy limit order
#%                       - NOTE: Execution of a sell (buy)
#%                               limit order corresponds to
#%                               a buyer-(seller-) initiated
#%                               trade, i.e. a BUY (SELL) trade.
#%
#% ----------------------------------------------------------
#% Data Preparation - Message File
#
#% Trading hours (start & end)

#%% deal with the message data
#Remove observations outside the official trading hours
# ----------------------------------------------------------

#% Trading hours (start & end)
start_trad   = 9.5*60*60       # 9:30:00 in sec
                               # after midnight
end_trad     = 16*60*60        # 16:00:00 in sec
                               # after midnight
```

```
# Get index of observations
time_idx=(mess[:,0]>= start_trad) & (mess[:,0]<= end_trad)
mess = mess[time_idx,:]


##-----------------------------------------------------------
#% Note: As the rows of the message and orderbook file
#%       correspond to each other, the time index of
#%       the message file can also be used to 'cut'
#%       the orderbook file.
#
#
#% Check for trading halts
#% ----------------------------------------------------------
trade_halt_idx = np.where(mess[:,1] == 7)

if (np.size(trade_halt_idx)>0):
    print(['Data contains trading halt! Trading halt, '+
    'quoting resume, and resume of trading indices in tradeHaltIdx'])
else:
    print('No trading halts detected.')
#
#
#%  When trading halts, a message of type '7' is written into the
#%  'message' file. The corresponding price and trade direction
#%  are set to '-1' and all other properties are set to '0'.
#%  Should the resume of quoting be indicated by an additional
#%  message in NASDAQ's Historical TotalView-ITCH files, another
#%  message of type '7' with price '0' is added to the 'message'
#%  file. Again, the trade direction is set to '-1' and all other
#%  fields are set to '0'.
#%  When trading resumes a message of type '7' and
#%  price '1' (Trade direction '-1' and all other
#%  entries '0') is written to the 'message' file. For messages
#%  of type '7', the corresponding order book rows contain a
#%  duplication of the preceding order book state. The reason
#%  for the trading halt is not included in the output.
#%
#%    Example: Stylized trading halt messages in 'message' file.
#%
#%    Halt:     36023 | 7 | 0 | 0 | -1 | -1
#%            ...
#%    Quoting:    36323  | 7 | 0 | 0 | 0  | -1
#%            ...
#%    Resume Trading:  36723   | 7 | 0 | 0 | 1  | -1
#%            ...
#%    The vertical bars indicate the different columns in the
#%    message file.
#
#% Set Bounds for Intraday Intervals
#
#% Define interval length
```

```python
freq = 6.5*3600/(5*60)+1  # Interval length in sec, according to the python do not include the endpoint
                          # so add 1 in the last

time_interval=60*6.5/(freq-1)

# Set interval bounds
bounds = np.linspace(start_trad,end_trad,freq,endpoint=True)

# Number of intervals
bl = np.size(bounds,0)

# Indices for intervals
bound_idx = np.zeros([bl,1])



k1 = 0
for k2 in range(0,np.size(mess,0)):
    if mess[k2,0] >= bounds[k1]:
        bound_idx[k1,0] = k2
        k1 = k1+1
bound_idx[bl-1]=mess[len(mess)-1,0]


#
#% Plot - Number of Executions and Trade Volume by Interval
#
#% Note: Difference between trades and executions
#%
#%      The LOBSTER output records limit order executions
#%      and not what one might intuitively consider trades.
#%
#%      Imagine a volume of 1000 is posted at the best ask
#%      price. Further, an incoming market buy order of
#%      volume 1000 is executed against the quote.
#%
#%      The LOBSTER output of this trade depends on the
#%      composition of the volume at the best ask price.
#%      Take the following two scenarios with the best ask
#%      volume consisting of ...
#%      (a) 1 sell limit order with volume 1000
#%      (b) 5 sell limit orders with volume 200 each
#%          (ordered according to time of submission)
#%
#%      The LOBSTER output for case ...
#%      (a) shows one execution of volume 1000. If the
#%          incoming market order is matched with one
#%          standing limit order, execution and trade
#%          coincide.
#%      (b) shows 5 executions of volume 200 each with the
#%          same time stamp. The incoming order is matched
#%          with 5 standing limit orders and triggers 5
#%          executions.
```

```python
#%
#%       Bottom line:
#%       LOBSTER records the exact limit orders against
#%       which incoming market orders are executed. What
#%       might be called 'economic' trade size has to be
#%       inferred from the executions.

#% Collection matrix
trades_info = np.zeros([bl-1,4])
#    % Note: Number visible executions, volume visible
#    %       trades, number hidden executions,
#    %       volume hidden trades


for k1 in range(0,bl-1):


    temp = mess[int(bound_idx[k1]+1):int(bound_idx[k1+1]),[1,3]]

    temp_vis = temp[temp[:,0]==4,1]    # Visible


    #% Hidden
    temp_hid = temp[temp[:,0]==5,1];

    # Collect information
    trades_info[k1,:] = [np.size(temp_vis,0), np.sum(temp_vis),np.size(temp_hid,0), np.sum(temp_hid)]

    del temp, temp_vis, temp_hid

#%% plot the data
#Plot number of executions
#--------------------------------------------------------------------------------

%matplotlib inline
fig, ax = plt.subplots()
ind=np.arange(np.size(trades_info,0))
width=1
color=["red","blue"]
   #% Visible ...
ax.bar(ind,trades_info[:,0],width=width, color=color[0],label="Visible",alpha=0.7)
#       title({[ticker ' // ' ...
#           datestr(datenum(demoDate),'yyyy-mmm-dd')] ...
#           ['Number of Executions per ' ...
#           num2str(freq./60) ' min Interval ']});
ax.set_xlabel('Interval')
ax.set_ylabel('Number of Executions')
ax.set_title(ticker+"@"+str(demo_date[0])+"-"+str(demo_date[1])+
"-"+str(demo_date[2])+"\nNumber of Executions per "+str(time_interval)+" minutes interval")
ax.bar(ind,-trades_info[:,2],width=width,color=color[1],label="Hidden");
ax.legend(loc="upper center")
plt.savefig(ticker+"_num_exec.png")
```

```
#-------------------------------------------------------------------------------
#plot the volume of traders
#-------------------------------------------------------------------------------
fig, ax = plt.subplots()
ind=np.arange(np.size(trades_info,0))
width=1
color=["red","blue"]
   #% Visible ...
ax.bar(ind,trades_info[:,1]/100,width=width, color=color[0],label="visible",alpha=0.7)

ax.set_xlabel('Interval')
ax.set_ylabel('Number of Trades Trades (X100 shares)')
ax.set_title(ticker+"@"+str(demo_date[0])+"-"+str(demo_date[1])+
"-"+str(demo_date[2])+"\nVolume of trades per "+str(time_interval)+" minutes interval")
ax.bar(ind,-trades_info[:,3]/100,width=width,color=color[1],label="Hidden");
ax.legend(loc="upper center")
plt.savefig(ticker+"_num_trade.png")
plt.show()

t=time.time()
book = np.array(pd.read_csv(path+name_book,dtype ="float64"))
print("The time for reading the CSV file",time.time()-t)
book = book[time_idx,:]
book[:,::2]=book[:,::2]/10000

#%% plot the snapshot of the limit order book
#-------------------------------------------------------------------------------
#select a random event to show
event_idx= np.random.randint(0, len(book))# note that the randint will not generate the last value

ask_price_pos=list(range(0,lvl*4,4))

# Note: Pick a randmom row/ event from the order book.
# position of variables in the book

ask_price_pos = list(range(0,lvl*4,4))

ask_vol_pos= [i+1 for i in ask_price_pos]

bid_price_pos=[i+2 for i in ask_price_pos]

bid_vol_pos=[i+1 for i in bid_price_pos]

vol= list(range(1,lvl*4,2))

max_price = book[event_idx, ask_price_pos[lvl-1]]+0.01
min_price=book[event_idx,bid_price_pos[lvl-1]]-0.01

max_vol=max(book[event_idx,vol])

mid=0.5*(sum(book[event_idx,[0,2]],2))
```

```
#%%plot the Snapshot of the Limit Order Book
#-------------------------------------------------------------------------------
plt.figure()
#ask price
color=["red","blue"]
y_pos=np.arange(11,21)
y_value=book[event_idx,ask_vol_pos]
plt.barh(y_pos, y_value,alpha=0.7,color=color[0],align="center",label="Ask")
#mid price
plt.plot([10,40],[10,10],'<g',markersize=10,fillstyle="full",label="Mid_price")
#bid price
y_pos=np.arange(0,10)
y_value=book[event_idx,bid_vol_pos][::-1]
plt.barh(y_pos,y_value,alpha=0.7,color=color[1],align="center",label="Bid")
#set style
y_pos=np.arange(0,21)
y_ticks=np.concatenate((book[event_idx,bid_price_pos][::-1],np.array([mid]),book[event_idx,ask_price_pos]),0)
plt.yticks(y_pos,y_ticks)
plt.xlabel('Volumne')
plt.title(ticker+"@"+str(demo_date[0])+"-"+str(demo_date[1])+
"-"+str(demo_date[2])+"\nLOB Snapshot -Time: "+str(mess[event_idx,0])+" Seconds")
plt.ylim([-1,21])
plt.legend()
plt.savefig(ticker+"_snapshot.png")

plt.show()

#%%plot the relative depth in the Limit Oeder Book
#-------------------------------------------------------------------------------



#% Relative volume ...

#% Ask
book_vol_ask = np.cumsum(book[event_idx,ask_vol_pos])
book_vol_ask = book_vol_ask/book_vol_ask[-1]

#% Bid
book_vol_bid = np.cumsum(book[event_idx,bid_vol_pos])
book_vol_bid = book_vol_bid/book_vol_bid[-1]

plt.figure()
#% Ask
plt.step(list(range(1,11)),book_vol_ask,color="g",label="Ask Depth")

plt.title(ticker+"@"+str(demo_date[0])+"-"+str(demo_date[1])+
"-"+str(demo_date[2])+"\nLOB Relative Depth -Time: "+str(mess[event_idx,0])+" Seconds")

plt.ylabel('% of Volume')
plt.xlabel('Level')

plt.xlim([1,10])
```

```
#Bid
plt.step(list(range(1,11)),-book_vol_ask,color="r",label="Bid Depth")

#y_pos=np.arange(0,21)
y_pos=np.linspace(-1,1,11)
plt.yticks(y_pos,[1,0.8,0.6,0.4,0.2,0,0.2,0.4,0.6,0.8,1])
plt.ylim([-1,1])
plt.savefig(ticker+"_depth.png")

plt.show()
```

1.Plot the order book types

In [ ]:

```
import time
order_type_list=[]
t=time.time()
for ticker_ind in range(5):
    order_type=[]
    for i in [1,2,3,4,5]:
        order_type.append(sum(data_mess_list[ticker_ind][:,1]==i))
    order_type_list.append(order_type)
print(time.time()-t)
```

In [ ]:

```
print(order_type_list[4])
```

In [ ]:

```
%matplotlib qt

import numpy as np
import matplotlib.pyplot as plt
# n_groups = 5

# means_men = (20, 35, 30, 35, 27)
# std_men = (2, 3, 4, 1, 2)

# means_women = (25, 32, 34, 20, 25)
# std_women = (3, 5, 2, 3, 3)

# fig, ax = plt.subplots()

# index = np.arange(n_groups)
# bar_width = 0.35

# opacity = 0.4
# error_config = {'ecolor': '0.3'}
```

```
# rects1 = plt.bar(index, means_men, bar_width,
#                  alpha=opacity,
#                  color='b',
#                  yerr=std_men,
#                  error_kw=error_config,
#                  label='Men')

# rects2 = plt.bar(index + bar_width, means_women, bar_width,
#                  alpha=opacity,
#                  color='r',
#                  yerr=std_women,
#                  error_kw=error_config,
#                  label='Women')

# plt.xlabel('Group')
# plt.ylabel('Scores')
# plt.title('Scores by group and gender')
# plt.xticks(index + bar_width, ('A', 'B', 'C', 'D', 'E'))
# plt.legend()

# plt.tight_layout()
# plt.show()

order_type_array=np.array(order_type_list)


n_groups=7.5
index = np.arange(n_groups,step=1.5)     # the x locations for the groups
ticker_list=['AAPL', 'AMZN', 'GOOG', 'INTC','MSFT']
color_list=['red','yellow','green','blue','darkmagenta']
type_list=['1:Order_book','2:Cancel_part','3:Delete_all','4:Execution_visible','5:Execution_hidden']

fig, ax = plt.subplots()

bar_width = 0.25

opacity = 0.6
error_config = {'ecolor': '0.3'}

rects1 = plt.bar(index, order_type_array[:,0], bar_width,
                 alpha=opacity,
                 color=color_list[0],
                 error_kw=error_config,
                 label=type_list[0])

rects2 = plt.bar(index + 1*bar_width, order_type_array[:,1], bar_width,
                 alpha=opacity,
                 color=color_list[1],
                 error_kw=error_config,
                 label=type_list[1])
```

```
rects3 = plt.bar(index + 2*bar_width, order_type_array[:,2], bar_width,
                 alpha=opacity,
                 color=color_list[2],
                 error_kw=error_config,
                 label=type_list[2])

rects4 = plt.bar(index + 3*bar_width, order_type_array[:,3], bar_width,
                 alpha=opacity,
                 color=color_list[3],
                 error_kw=error_config,
                 label=type_list[3])

rects5 = plt.bar(index + 4*bar_width, order_type_array[:,4], bar_width,
                 alpha=opacity,
                 color=color_list[4],
                 error_kw=error_config,
                 label=type_list[4])


plt.xlabel('Stock Ticker')
plt.ylabel('Numbers')
plt.title('Order Book Types')
plt.xticks(index + bar_width*2.5, ticker_list)
plt.yticks(np.arange(0, 700000,50000))
plt.legend()
plt.tight_layout()
plt.show()
```

2.Plot the arbitrage situation (bid high, ask low and no arbitrage)

Take the first stock which is AAPL as example

In [ ]:

```
data_order_reduced=data_order_list[0][(time_index_list[0]>= start_ind) & (time_index_list[0]<= end_ind)]
data_mess_reduced=data_mess_list[0][(time_index_list[0]>= start_ind) & (time_index_list[0]<= end_ind)]
time_index_reduced=time_index_list[0][(time_index_list[0]>= start_ind) & (time_index_list[0]<= end_ind)]
```

In [ ]:

```
first_ind=np.where(ask_low_time_list[0][1]==1)[0][0]
last_ind=np.where(time_index_reduced>time_index_reduced[first_ind]+5)[0][0]
print("first_ind:",first_ind)
print("last_ind:",last_ind)
```

In [ ]:

```
%matplotlib qt
time_index=time_index_reduced[first_ind:last_ind+1]
ask_price=data_order_reduced[first_ind:last_ind+1,0]
bid_price=data_order_reduced[first_ind:last_ind+1,2]
```

```
bid_price=data_order_reduced[first_ind:last_ind+1,2]
print(ask_pirce[1])
print(bid_price[1])
plt.plot(time_index,ask_price,'r.-',label="Ask Price")
plt.plot(time_index,bid_price,'b.-',label="Bid Price")

plt.xticks=time_index
plt.xlabel("Time")
plt.ylabel("Price")
plt.title("Ask Low Arbitrage Example")
plt.legend(loc='upper center')
plt.show()
```

In [ ]:

```
np.where(bid_high_time_list[0][1]==1)
```

In [ ]:

```
## the bid high case

first_ind=np.where(bid_high_time_list[0][1]==1)[0][20]
last_ind=np.where(time_index_list[0]>time_index_list[0][first_ind]+5)[0][0]
print("first_ind:",first_ind)
print("last_ind:",last_ind)

%matplotlib qt
time_index=time_index_list[0][first_ind:last_ind+1]
ask_price=data_order_list[0][first_ind:last_ind+1,0]
bid_price=data_order_list[0][first_ind:last_ind+1,2]
print(ask_pirce[1])
print(bid_price[1])
plt.plot(time_index,ask_price,'r.-',label="Ask Price")
plt.plot(time_index,bid_price,'b.-',label="Bid Price")

plt.xticks=time_index
plt.xlabel("Time")
plt.ylabel("Price")
plt.title("Bid High Arbitrage Example")
plt.legend(loc='upper center')
plt.show()
```

In [ ]:

```
## the no arbitrage case
first_ind=np.where(no_arbi_time_list[0][1]==1)[0][20]
last_ind=np.where(time_index_list[0]>time_index_list[0][first_ind]+5)[0][0]
print("first_ind:",first_ind)
print("last_ind:",last_ind)

%matplotlib qt
time_index=time_index_list[0][first_ind:last_ind+1]
ask_price=data_order_list[0][first_ind:last_ind+1,0]
```

```
bid_price=data_order_list[0][first_ind:last_ind+1,2]
print(ask_pirce[1])
print(bid_price[1])
plt.plot(time_index,ask_price,'r.-',label="Ask Price")
plt.plot(time_index,bid_price,'b.-',label="Bid Price")

plt.xticks=time_index
plt.xlabel("Time")
plt.ylabel("Price")
plt.title("No Arbitrage Example")
plt.legend(loc='upper center')
plt.show()
```

3.plot the statistical properties

1) cumulative distribution function for arrival time

In [ ]:

```
ticker_ind=2
data=data_mess_list[ticker_ind]
# we use the market order
data_order=data[(data[:,1]==4) | (data[:,1]==5)]

arrival_time=data_order[1:,0]-data_order[0:-1,0]
#delete the zero intra arrival time
arrival_time=arrival_time[arrival_time>0]
```

In [ ]:

```
mu_log=np.mean(np.log(arrival_time))
std_log=np.std(np.log(arrival_time))
data_log=np.random.lognormal(mu_log,std_log,arrival_time.shape)

mu_exp=np.mean(arrival_time)
data_exp=np.random.exponential(mu_exp,arrival_time.shape)

data_weibull=np.random.weibull(0.38,arrival_time.shape)
beta=np.var(arrival_time)/np.mean(arrival_time)
alpha=np.mean(arrival_time)/beta
data_gamma=np.random.gamma(alpha,beta,arrival_time.shape)
```

In [ ]:

```
%matplotlib inline
import statsmodels.api as sm
from scipy.stats.kde import gaussian_kde

from scipy.interpolate import UnivariateSpline
from scipy.stats import lognorm
```

```
from scipy.stats import lognorm
ecdf = sm.distributions.ECDF(arrival_time,)
plt.xlim([0,10])
plt.plot(ecdf.x, ecdf.y,"b",label="Original data")

ecdf = sm.distributions.ECDF(data_log)
plt.xlim([0,10])
plt.plot(ecdf.x, ecdf.y,"g",label="Lognormal Distribution")

ecdf = sm.distributions.ECDF(data_exp)
plt.xlim([0,10])
plt.plot(ecdf.x, ecdf.y,"y",label="Exponential distribution")

ecdf = sm.distributions.ECDF(data_weibull)
plt.xlim([0,10])
plt.plot(ecdf.x, ecdf.y,"r",label="Weibull distribution")


ecdf = sm.distributions.ECDF(data_gamma)
plt.xlim([0,10])
plt.plot(ecdf.x, ecdf.y,"purple",label="Gamma distribution")


plt.xlabel("Intra-arrival time")
plt.ylabel("Probability")
plt.legend(loc="lower right")
plt.title("Cumulative distribution function of order arrival time")
plt.show()
```

1) loop for all stocks

In [ ]:

```
f, axarr = plt.subplots(2, 2,figsize=(13, 13))
for ticker_ind in range(1,5):
    data=data_mess_list[ticker_ind]
    # we use the market order
    data_order=data[(data[:,1]==4) | (data[:,1]==5)]

    arrival_time=data_order[1:,0]-data_order[0:-1,0]
    #delete the zero intra arrival time
    arrival_time=arrival_time[arrival_time>0]
    mu_log=np.mean(np.log(arrival_time))
    std_log=np.std(np.log(arrival_time))
    data_log=np.random.lognormal(mu_log,std_log,arrival_time.shape)

    mu_exp=np.mean(arrival_time)
    data_exp=np.random.exponential(mu_exp,arrival_time.shape)

    data_weibull=np.random.weibull(0.38,arrival_time.shape)
    beta=np.var(arrival_time)/np.mean(arrival_time)
    alpha=np.mean(arrival_time)/beta
```

```
    data_gamma=np.random.gamma(alpha,beta,arrival_time.shape)
    ecdf = sm.distributions.ECDF(arrival_time,)


    axarr[int((ticker_ind-1)/2),(ticker_ind+1)%2].set_xlim([0,10])
    axarr[int((ticker_ind-1)/2),(ticker_ind+1)%2].plot(ecdf.x, ecdf.y,"b",label="Original data")

    ecdf = sm.distributions.ECDF(data_log)
    axarr[int((ticker_ind-1)/2),(ticker_ind+1)%2].set_xlim([0,10])
    axarr[int((ticker_ind-1)/2),(ticker_ind+1)%2].plot(ecdf.x, ecdf.y,"g",label="Lognormal Distribution")

    ecdf = sm.distributions.ECDF(data_exp)
    axarr[int((ticker_ind-1)/2),(ticker_ind+1)%2].set_xlim([0,10])
    axarr[int((ticker_ind-1)/2),(ticker_ind+1)%2].plot(ecdf.x, ecdf.y,"y",label="Exponential distribution")

    ecdf = sm.distributions.ECDF(data_weibull)
    axarr[int((ticker_ind-1)/2),(ticker_ind+1)%2].set_xlim([0,10])
    axarr[int((ticker_ind-1)/2),(ticker_ind+1)%2].plot(ecdf.x, ecdf.y,"r",label="Weibull distribution")


    ecdf = sm.distributions.ECDF(data_gamma)
    axarr[int((ticker_ind-1)/2),(ticker_ind+1)%2].set_xlim([0,10])
    axarr[int((ticker_ind-1)/2),(ticker_ind+1)%2].plot(ecdf.x, ecdf.y,"purple",label="Gamma distribution")


    axarr[int((ticker_ind-1)/2),(ticker_ind+1)%2].set_xlabel("Intra-arrival time")
    axarr[int((ticker_ind-1)/2),(ticker_ind+1)%2].set_ylabel("Probability")
    axarr[int((ticker_ind-1)/2),(ticker_ind+1)%2].legend(loc="lower right")
    axarr[int((ticker_ind-1)/2),(ticker_ind+1)%2].set_title("Cumulative distribution function of \n order arrival time  for stock "+ticker_list[ticker_ind
])

plt.savefig('arrival_time.png', bbox_inches='tight')
plt.show()
```

2) volume

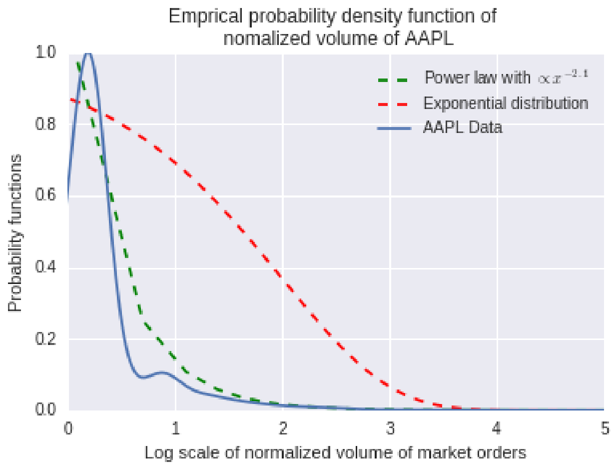
In [6]:

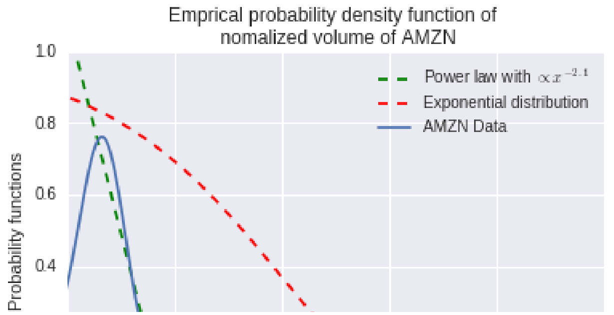
```
%matplotlib inline
from scipy.interpolate import UnivariateSpline
from scipy.stats import lognorm
import seaborn as sns
ticker_ind=0
x=np.linspace(0,50,1000)
y=x**(-2.1)/500
plt.plot(np.log(x)+3,y,"g--",label="Power law with $\propto x^{-2.1}$")
y_exp=np.exp(-x)
plt.plot(np.log(x)+2,y_exp,"r--",label="Exponential distribution")
data=data_mess_list[ticker_ind]

data_market=data[(data[:,1]==4) | (data[:,1]==5)]
```

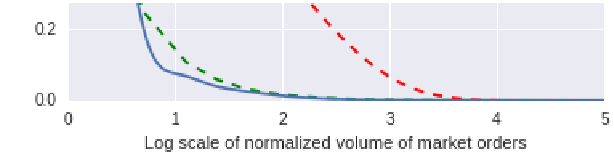
```
data_order=data[data[:,1]==1]
mean_market=np.mean(data_market[:,3])
mean_order=np.mean(data_order[:,3])

vol_market_scale=data_market[:,3]/mean_market
vol_order_scale=data_order[:,3]/mean_order
Se_u=pd.Series(np.log(vol_market_scale))
Se_u.plot(kind="kde",label=ticker_list[ticker_ind]+" Data")

plt.xlim([0,5])
plt.ylim([0,1])
plt.legend()
plt.xlabel("Log scale of normalized volume of market orders")
plt.ylabel("Probability functions")
plt.title("Emprical probability density function of \n nomalized volume of "+ticker_list[ticker_ind])
plt.savefig("volume_AAPL.png")
plt.show()

ticker_ind=1
x=np.linspace(0,50,1000)
y=x**(-2.1)/500
plt.plot(np.log(x)+3,y,"g--",label="Power law with $\propto x^{-2.1}$")
y_exp=np.exp(-x)
plt.plot(np.log(x)+2,y_exp,"r--",label="Exponential distribution")
data=data_mess_list[ticker_ind]

data_market=data[(data[:,1]==4) | (data[:,1]==5)]
data_order=data[data[:,1]==1]
mean_market=np.mean(data_market[:,3])
mean_order=np.mean(data_order[:,3])

vol_market_scale=data_market[:,3]/mean_market
vol_order_scale=data_order[:,3]/mean_order
Se_u=pd.Series(np.log(vol_market_scale))
Se_u.plot(kind="kde",label=ticker_list[ticker_ind]+" Data")

plt.xlim([0,5])
plt.ylim([0,1])
plt.legend()
plt.xlabel("Log scale of normalized volume of market orders")
plt.ylabel("Probability functions")
plt.title("Emprical probability density function of \n nomalized volume of "+ticker_list[ticker_ind])
plt.savefig("volume_AMZN.png")
plt.show()


ticker_ind=2
x=np.linspace(0,50,1000)
y=x**(-2.1)/500
plt.plot(np.log(x)+3,y,"g--",label="Power law with $\propto x^{-2.1}$")
y_exp=np.exp(-x)
plt.plot(np.log(x)+2,y_exp,"r--",label="Exponential distribution")
data=data_mess_list[ticker_ind]
```

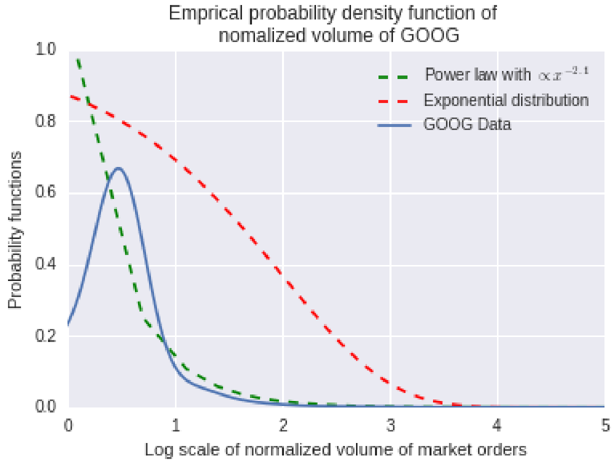
```
data_market=data[(data[:,1]==4) | (data[:,1]==5)]
data_order=data[data[:,1]==1]
mean_market=np.mean(data_market[:,3])
mean_order=np.mean(data_order[:,3])

vol_market_scale=data_market[:,3]/mean_market
vol_order_scale=data_order[:,3]/mean_order

Se_u=pd.Series(np.log(vol_market_scale))
Se_u.plot(kind="kde",label=ticker_list[ticker_ind]+" Data")
plt.xlim([0,5])
plt.ylim([0,1])
plt.legend()
plt.xlabel("Log scale of normalized volume of market orders")
plt.ylabel("Probability functions")
plt.title("Emprical probability density function of \n nomalized volume of "+ticker_list[ticker_ind])
plt.savefig("volume_INTC.png")
plt.show()


ticker_ind=3
x=np.linspace(0,50,1000)
y=x**(-2.1)/500
plt.plot(np.log(x)+3,y,"g--",label="Power law with $\propto x^{-2.1}$")
y_exp=np.exp(-x)
plt.plot(np.log(x)+2,y_exp,"r--",label="Exponential distribution")
data=data_mess_list[ticker_ind]

data_market=data[(data[:,1]==4) | (data[:,1]==5)]
data_order=data[data[:,1]==1]
mean_market=np.mean(data_market[:,3])
mean_order=np.mean(data_order[:,3])

vol_market_scale=data_market[:,3]/mean_market
vol_order_scale=data_order[:,3]/mean_order

Se_u=pd.Series(np.log(vol_market_scale))
Se_u.plot(kind="kde",label=ticker_list[ticker_ind]+" Data")
plt.xlim([0,5])
plt.ylim([0,1])
plt.legend()
plt.xlabel("Log scale of normalized volume of market orders")
plt.ylabel("Probability functions")
plt.title("Emprical probability density function of \n nomalized volume of "+ticker_list[ticker_ind])
plt.savefig("volume_GOOG.png")
plt.show()

ticker_ind=4
x=np.linspace(0,50,1000)
y=x**(-2.1)/500
plt.plot(np.log(x)+3,y,"g--",label="Power law with $\propto x^{-2.1}$")
y_exp=np.exp(-x)
```

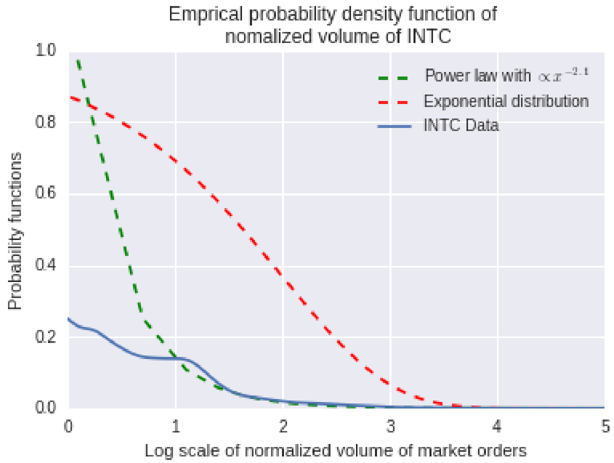
```
y_exp=np.exp(-x)
plt.plot(np.log(x)+2,y_exp,"r--",label="Exponential distribution")
data=data_mess_list[ticker_ind]

data_market=data[(data[:,1]==4) | (data[:,1]==5)]
data_order=data[data[:,1]==1]
mean_market=np.mean(data_market[:,3])
mean_order=np.mean(data_order[:,3])

vol_market_scale=data_market[:,3]/mean_market
vol_order_scale=data_order[:,3]/mean_order

Se_u=pd.Series(np.log(vol_market_scale))
Se_u.plot(kind="kde",label=ticker_list[ticker_ind]+" Data")

plt.xlim([0,5])
plt.ylim()
plt.legend()
plt.xlabel("Log scale of normalized volume of market orders")
plt.ylabel("Probability functions")
plt.title("Emprical probability density function of \n nomalized volume of "+ticker_list[ticker_ind])
plt.savefig("volume_MSFT.png")
plt.show()
```

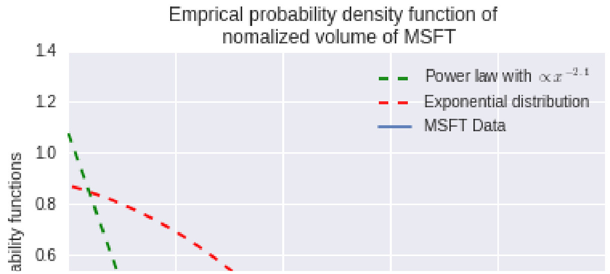


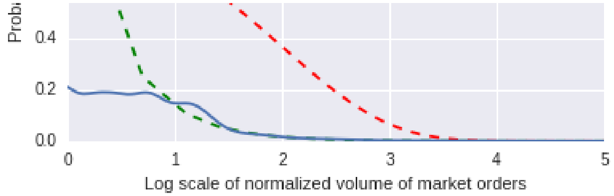



0.2
0.0
0
1
2
3
4
5
Log scale of normalized volume of market orders


Emprical probability density function of
nomalized volume of GOOG
Power law with $\propto x^{-2.1}$
Exponential distribution
GOOG Data
Probability functions
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
Log scale of normalized volume of market orders


Emprical probability density function of
nomalized volume of INTC
Power law with $\propto x^{-2.1}$
Exponential distribution
INTC Data
Probability functions
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
Log scale of normalized volume of market orders


Emprical probability density function of
nomalized volume of MSFT
Power law with $\propto x^{-2.1}$
Exponential distribution
MSFT Data
1.4
1.2
1.0
0.8
0.6



3) Intraday seasonality

observe the volume during the whole day under 5 minutes time bins. show the result of seasonality

In [13]:

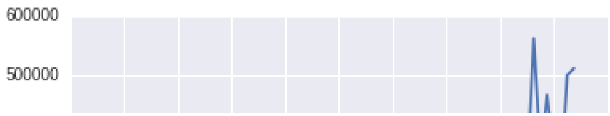
```
ticker_ind=0
data_mess=data_mess_list[ticker_ind]
data_mess_limit=data_mess[data_mess[:,1]==1,:]
```

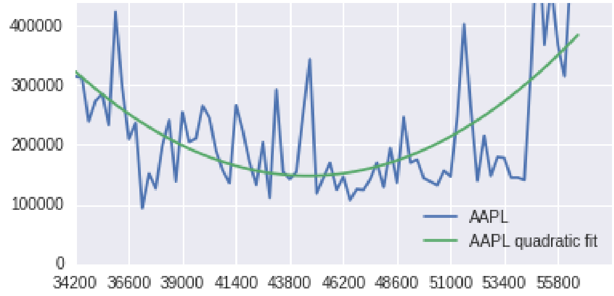
In [14]:

```
# calute the volume of limit order book in each time interval

time_interval=np.linspace(data_mess_limit[:,0].min(),data_mess_limit[:,0].max(),78)
vol=0
vol_time=[]
j=1

for i in range(len(data_mess_limit)):
    if  data_mess_limit[i,0]<=time_interval[j]:
        vol=vol+data_mess_limit[i,3]
    else:
        j=j+1
        vol_time.append(vol)
        vol=data_mess_limit[i,3]
```

In [15]:

```
# plot the quadratic fit and vol_time
x=range(76)
plt.plot(x,vol_time,label=ticker_list[ticker_ind])
qua_fit=np.poly1d(np.polyfit(x, vol_time, 2))(x)
plt.plot(x,qua_fit,label=ticker_list[ticker_ind]+" quadratic fit")
plt.legend(loc="lower right")
xticks=np.arange(34200,57600,2400)
plt.xticks(x[::8],xticks)
plt.show()
```



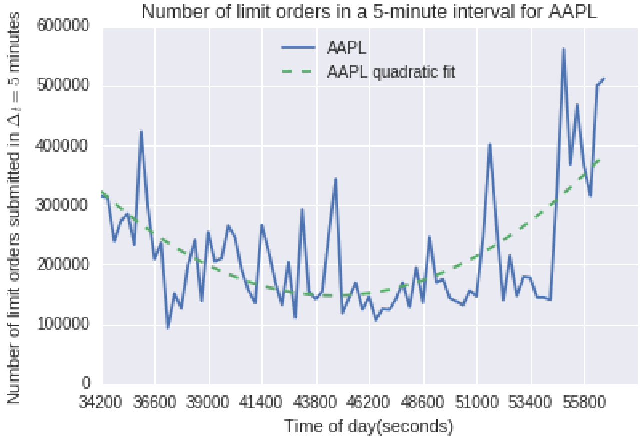


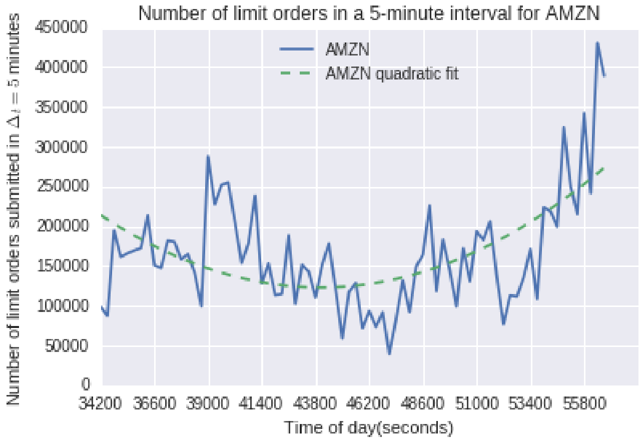
loop for all stocks

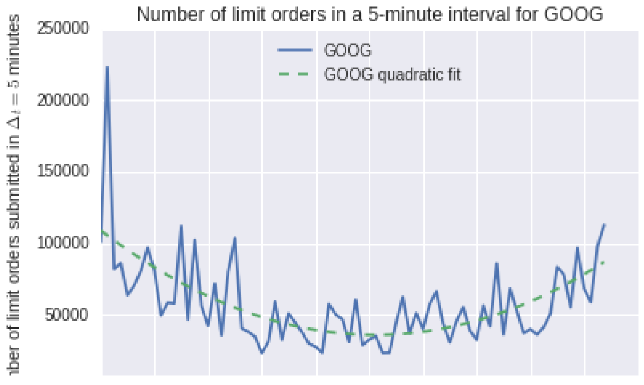
In [7]:

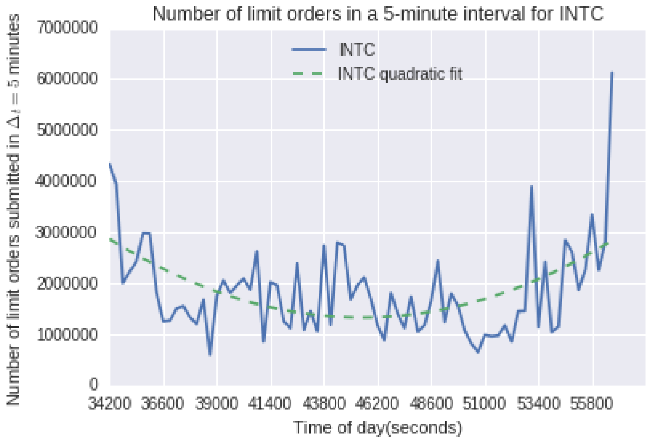
```
# for limit order


%matplotlib inline

import seaborn as sns

for ticker_ind in range(0,5):

    data_mess=data_mess_list[ticker_ind]
    data_mess_limit=data_mess[data_mess[:,1]==1,:]
    # calute the volume of limit order book in each time interval

    time_interval=np.linspace(data_mess_limit[:,0].min(),data_mess_limit[:,0].max(),78)
    vol=0
    vol_time=[]
    j=1

    for i in range(len(data_mess_limit)):
        if  data_mess_limit[i,0]<=time_interval[j]:
            vol=vol+data_mess_limit[i,3]
        else:
            j=j+1
            vol_time.append(vol)
            vol=data_mess_limit[i,3]
    # plot the quadratic fit and vol_time
    x=range(76)
    plt.plot(x,vol_time,"+-",label=ticker_list[ticker_ind])
    qua_fit=np.poly1d(np.polyfit(x, vol_time, 2))(x)
    plt.plot(x,qua_fit,"--",label=ticker_list[ticker_ind]+" quadratic fit")
    plt.legend(loc="upper center")
    xticks=np.arange(34200,57600,2400)
    plt.xticks(x[::8],xticks)
    plt.title("Number of limit orders in a 5-minute interval for "+ticker_list[ticker_ind])
    plt.xlabel("Time of day(seconds)")
```

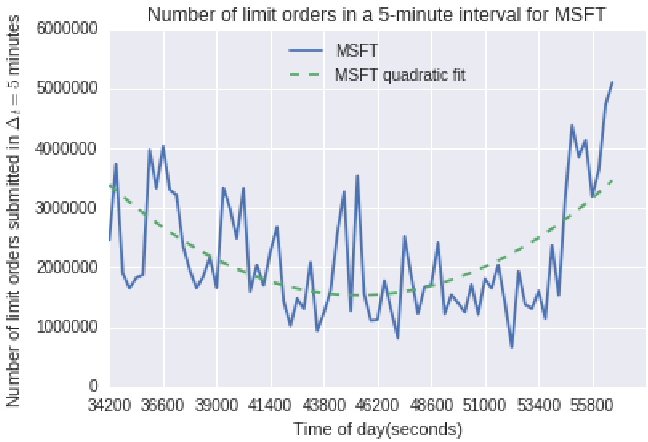
```
    plt.ylabel("Number of limit orders submitted in $\Delta_t=5$ minutes")
    plt.savefig(ticker_list[ticker_ind]+"_limit_vol_time.png",bbox_inches='tight')

    plt.show()
```







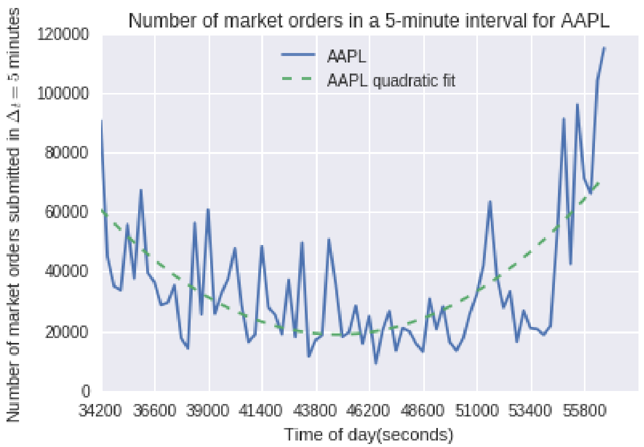


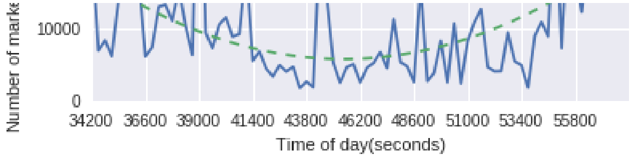


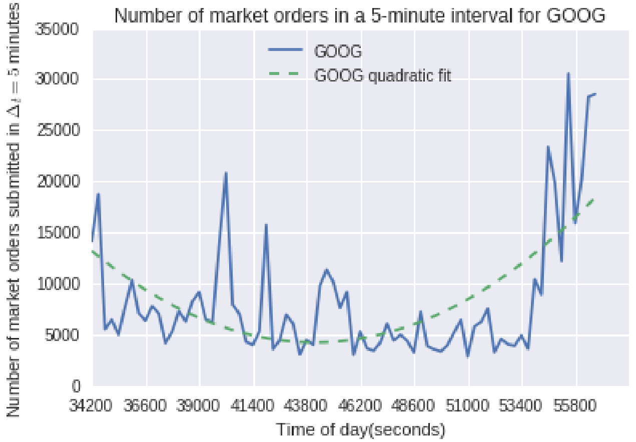
In [10]:

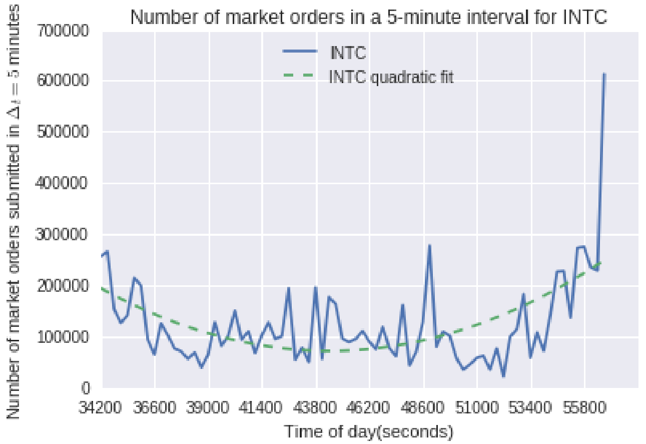
```
# market order

%matplotlib inline
import seaborn as sns

for ticker_ind in range(0,5):

    data_mess=data_mess_list[ticker_ind]
    data_mess_market=data_mess[(data_mess[:,1]==4) | (data_mess[:,1]==5),:]
    # calute the volume of limit order book in each time interval

    time_interval=np.linspace(data_mess_market[:,0].min(),data_mess_market[:,0].max(),78)
    vol=0
```

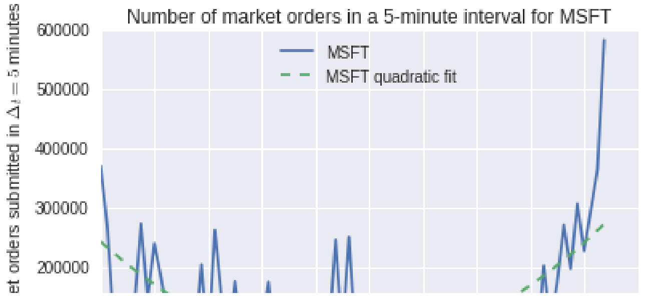
```
    vol=0
    vol_time=[]
    j=1

    for i in range(len(data_mess_market)):
        if  data_mess_market[i,0]<=time_interval[j]:
            vol=vol+data_mess_market[i,3]
        else:
            j=j+1
            vol_time.append(vol)
            vol=data_mess_market[i,3]
    # plot the quadratic fit and vol_time
    x=range(76)
    plt.plot(x,vol_time,"+-",label=ticker_list[ticker_ind])
    qua_fit=np.poly1d(np.polyfit(x, vol_time, 2))(x)
    plt.plot(x,qua_fit,"--",label=ticker_list[ticker_ind]+" quadratic fit")
    plt.legend(loc="upper center")
    xticks=np.arange(34200,57600,2400)
    plt.xticks(x[::8],xticks)
    plt.title("Number of market orders in a 5-minute interval for "+ticker_list[ticker_ind])
    plt.xlabel("Time of day(seconds)")
    plt.ylabel("Number of market orders submitted in $\Delta_t=5$ minutes")
    plt.savefig(ticker_list[ticker_ind]+"_market_vol_time.png",bbox_inches='tight')
    plt.show()
```

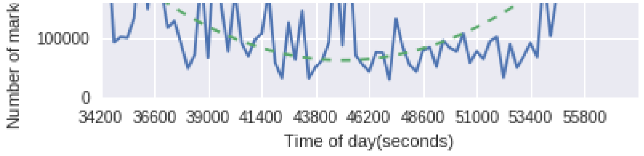



Number of market orders submitted in $\Delta_t = 5$ minutes
Time of day(seconds)


Number of market orders in a 5-minute interval for GOOG
GOOG
GOOG quadratic fit
Number of market orders submitted in $\Delta_t = 5$ minutes
Time of day(seconds)


Number of market orders in a 5-minute interval for INTC
INTC
INTC quadratic fit
Number of market orders submitted in $\Delta_t = 5$ minutes
Time of day(seconds)


Number of market orders in a 5-minute interval for MSFT
MSFT
MSFT quadratic fit



4) average shape of the order books

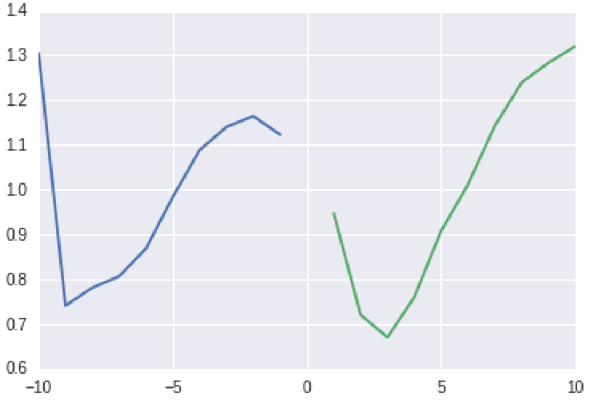
find the total volume for all each price level and see the volume trend based on the price levels

In [18]:

```
### 4) average shape of the order books
%matplotlib inline
import seaborn as sns

ticker_ind=1
data_mess=data_mess_list[ticker_ind]
data_order=data_order_list[ticker_ind]
data_order_limit_ask_vol=data_order[data_mess[:,1]==1,1:40:4]
data_order_limit_bid_vol=data_order[data_mess[:,1]==1,3:40:4]

vol_ask=np.sum(data_order_limit_ask_vol,axis=0)/np.mean(np.sum(data_order_limit_ask_vol,axis=0))
vol_bid=np.sum(data_order_limit_bid_vol,axis=0)/np.mean(np.sum(data_order_limit_bid_vol,axis=0))
plt.plot(list(range(-10,0)),vol_bid)
plt.plot(list(range(1,11)),vol_ask)
```

Out[18]:

```
[<matplotlib.lines.Line2D at 0x7f69d42b24a8>]
```



loop the stocks

In [15]:

```
marker_list=["s","D","^","8"]
color_list=["g","b","r","y"]
for ticker_ind in range(0,5):
    data_mess=data_mess_list[ticker_ind]
    data_order=data_order_list[ticker_ind]
    data_order_limit_ask_vol=data_order[:,1:40:4]
    data_order_limit_bid_vol=data_order[:,3:40:4]

    vol_ask=np.sum(data_order_limit_ask_vol,axis=0)/np.mean(np.sum(data_order_limit_ask_vol,axis=0))
    vol_bid=np.sum(data_order_limit_bid_vol,axis=0)/np.mean(np.sum(data_order_limit_bid_vol,axis=0))
    plt.plot(list(range(-10,0)),vol_bid,
             "--",marker=marker_list[ticker_ind-1],color=color_list[ticker_ind-1],label=
            ticker_list[ticker_ind])
    plt.plot(list(range(1,11)),vol_ask,"--",marker=marker_list[ticker_ind-1],color=color_list[ticker_ind-1])
plt.ylim([0.6,1.6])
plt.legend(loc="upper right")
plt.title("Average quantity offered in the market order book")
vol_bid=np.sum(data_order_limit_bid_vol,axis=0)/np.mean(np.sum(data_order_limit_bid_vol,axis=0))

plt.xlabel("Price level of limit orders (negative axis : bids ; positive axis : asks)")
plt.ylabel("Average numbers of shares(Normalized by mean)")
plt.savefig("level_quantity.png",bbox_inches='tight')
plt.show()
```

5) placement of orders

In [20]:

```
ticker_ind=2
data_mess=data_mess_list[ticker_ind]
data_order=data_order_list[ticker_ind]

data_mess_limit=data_mess[data_mess[:,1]==1,:]
```

```
data_order_limit=data_order[data_mess[:,1]==1,:]
```

In [21]:

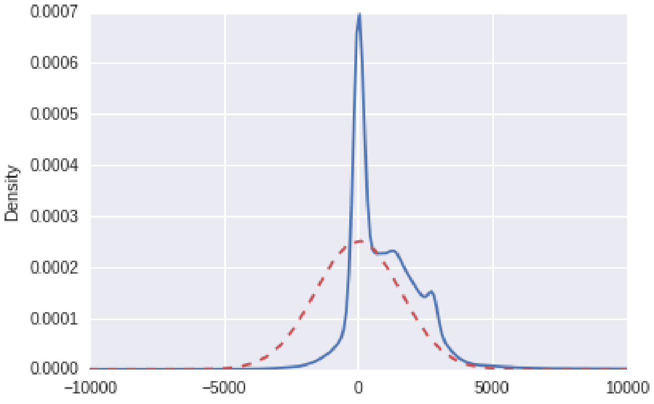
```
spread_list=[]
for i in range(1,len(data_mess_limit)):
    if data_mess_limit[i,5]==-1:
        spread=data_mess_limit[i,4]-data_order_limit[i-1,0]
    else:
        spread=data_order_limit[i-1,2]-data_mess_limit[i,4]
    spread_list.append(spread)
```

In [25]:

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.mlab as mlab
import math

Se_u=pd.Series(np.array(spread_list))
Se_u.plot(kind="kde",label=ticker_list[ticker_ind]+" Data")
mu = 0
variance = np.var(spread_list)
sigma = math.sqrt(variance)
x = np.linspace(min(spread_list), max(spread_list), 100)
plt.plot(x,mlab.normpdf(x, mu, sigma),"r--",label="Gaussian")
plt.xlim([-10000,10000])
```

Out[25]:

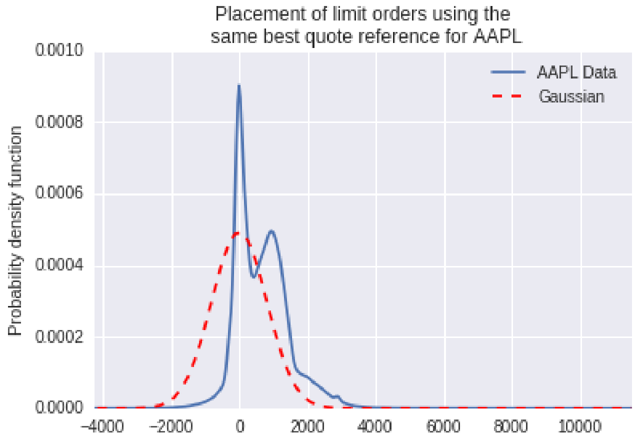
```
(-10000, 10000)
```



loop for all stocks

In [16]:

```
for ticker_ind in range(0,5):
    data_mess=data_mess_list[ticker_ind]
    data_order=data_order_list[ticker_ind]

    data_mess_limit=data_mess[data_mess[:,1]==1,:]
    data_order_limit=data_order[data_mess[:,1]==1,:]

    spread_list=[]
    for i in range(1,len(data_mess_limit)):
        if data_mess_limit[i,5]==-1:
            spread=data_mess_limit[i,4]-data_order_limit[i-1,0]
        else:
            spread=data_order_limit[i-1,2]-data_mess_limit[i,4]
        spread_list.append(spread)


    import seaborn as sns
    import matplotlib.pyplot as plt
    import numpy as np
    import matplotlib.mlab as mlab
    import math
    Se_u=pd.Series(np.array(spread_list))
    Se_u.plot(kind="kde",label=ticker_list[ticker_ind]+" Data")
    mu = 0
    variance = np.var(spread_list)
    sigma = math.sqrt(variance)
    x = np.linspace(min(spread_list), max(spread_list), 100)
    plt.plot(x,mlab.normpdf(x, mu, sigma),"r--",label="Gaussian")
    plt.xlim([min(spread_list)*0.8,max(spread_list)*0.8])
    plt.legend(loc="upper right")
    plt.title("Placement of limit orders using the\n same best quote reference for "+ticker_list[ticker_ind])
    plt.xlabel("Price diference")
    plt.ylabel("Probability density function")
    plt.savefig(ticker_list[ticker_ind]+"_placement.png",bbox_inches='tight')
    plt.show()
```

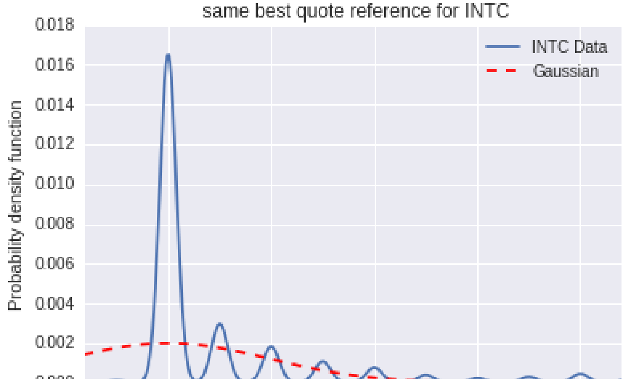


Price diference

Placement of limit orders using the
same best quote reference for AMZN

Placement of limit orders using the
same best quote reference for GOOG

Placement of limit orders using the
same best quote reference for INTC



Summary

In [76]:

```
#time series split
#%%-----------------------------------------------------------------------------------------------

ticker_ind=0
size=100000
random_ratio=0.6
# combine the feature and response array to random sample
total_array=np.concatenate((feature_array_list[ticker_ind],response_reduced_list[ticker_ind]),axis=1)[:size,:]

total_array=total_array[random_choice(list(range(size)),int(size*random_ratio)),:]


train_num_index=int(len(total_array)*0.9)

print("total array shape:",total_array.shape)

#split the data to train and test data set
train_x=total_array[:train_num_index,:134]
test_x=total_array[train_num_index:,:134]
train_y=total_array[:train_num_index,134]
test_y=total_array[train_num_index:,134]


# the y data need to reshape to size (n,) not (n,1)
test_y=test_y.reshape(len(test_y),)
train_y=train_y.reshape(len(train_y),)
```

```
print("train_x shape:",train_x.shape)
print("test_x shape:",test_x.shape)
print("test_y shape:",test_y.shape)
print("train_y shape:",train_y.shape)
# scale data
#%%

# can use the processing.scale function to scale the data
from sklearn import preprocessing
# note that we need to transfer the data type to float
# remark: should use data_test=data_test.astype('float'),very important !!!!
# use scale for zero mean and one std
scaler = preprocessing.StandardScaler().fit(train_x)


train_x_scale=scaler.transform(train_x)
test_x_scale=scaler.transform(test_x)

print(np.mean(train_x_scale,0))
print(np.mean(test_x_scale,0))

# -*- coding: utf-8 -*-

# set the sample weights for the training model
sample_weights=[]
ratio=len(train_y)/sum(train_y==1)/10
for i in range(len(train_x)):
    if train_y[i]==0:
        sample_weights.append(1)
    else: sample_weights.append(ratio)


#----------------
# logistic l1
#-----------------


from sklearn import linear_model

# set the sample weights for the training model
sample_weights=[]
ratio=len(train_y)/sum(train_y==1)/10
for i in range(len(train_x)):
    if train_y[i]==0:
        sample_weights.append(1)
    else: sample_weights.append(ratio)

        # set the random state to make sure that each time get the same results

time_logistic=time.time()
clf = linear_model.LogisticRegression(C=1, penalty='l1', tol=1e-6,random_state= 987612345)
clf.fit(train_x_scale,train_y)
```

```
time_logistic=time.time()-time_logistic

print(time_logistic)

# test the training error
predict_y_logistic =np.array(clf.predict(train_x_scale))
print("train_accuracy is:",sum(predict_y_logistic==train_y)/len(train_y))

# test the score for the train data
from sklearn.metrics import (brier_score_loss, precision_score, recall_score,
                             f1_score)
precision= precision_score(predict_y_logistic,train_y)
recall = recall_score(predict_y_logistic,train_y)
f1=f1_score(predict_y_logistic,train_y)
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)

# define a function to prefict the result by threshold
# note: logistic model will return two probability
def predict_threshold(predict_proba, threshold):
    res=[]
    for i in range(len(predict_proba)):
        res.append(int(predict_proba[i][1]>threshold))
    return res
t=time.time()
predict_y_test_proba =np.array(clf.predict_proba(test_x_scale))
print("test time is:", time.time()-t)

predict_y_test=predict_threshold(predict_y_test_proba,0.5)

# test the score for the train data
from sklearn.metrics import (precision_score, recall_score,
                             f1_score)
print("accuracy is:",sum(predict_y_test==test_y)/len(test_y))
precision= precision_score(predict_y_test,test_y)
recall = recall_score(predict_y_test,test_y)
f1=f1_score(predict_y_test,test_y)
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)

%matplotlib inline
## draw chart for the cross table

from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(2)
    plt.xticks(tick_marks, [0,1])
```

```
    plt.yticks(tick_marks, [0,1])
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


# Compute confusion matrix
cm = confusion_matrix(test_y, predict_y_test)
np.set_printoptions(precision=2)
print('Confusion matrix, without normalization')
print(cm)
plt.figure()
plot_confusion_matrix(cm)
plt.show()

#----------------
# logistic l2
#-----------------


from sklearn import linear_model

# set the sample weights for the training model
sample_weights=[]
ratio=len(train_y)/sum(train_y==1)/10
for i in range(len(train_x)):
    if train_y[i]==0:
        sample_weights.append(1)
    else: sample_weights.append(ratio)

        # set the random state to make sure that each time get the same results

time_logistic=time.time()
clf = linear_model.LogisticRegression(C=1, penalty='l2', tol=1e-6,random_state= 987612345)
clf.fit(train_x_scale,train_y)
time_logistic=time.time()-time_logistic

print(time_logistic)

# test the training error
t=time.time()
predict_y_logistic =np.array(clf.predict(train_x_scale))
print("test time is:", time.time()-t)

print("train_accuracy is:",sum(predict_y_logistic==train_y)/len(train_y))

# test the score for the train data
from sklearn.metrics import (brier_score_loss, precision_score, recall_score,
                             f1_score)
precision= precision_score(predict_y_logistic,train_y)
recall = recall_score(predict_y_logistic,train_y)
f1=f1_score(predict_y_logistic,train_y)
print("precision is: \t %s" % precision)
```

```
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)

# define a function to prefict the result by threshold
# note: logistic model will return two probabilitiy
def predict_threshold(predict_proba, threshold):
    res=[]
    for i in range(len(predict_proba)):
        res.append(int(predict_proba[i][1]>threshold))
    return res
predict_y_test_proba =np.array(clf.predict_proba(test_x_scale))

predict_y_test=predict_threshold(predict_y_test_proba,0.5)

# test the score for the train data
from sklearn.metrics import (precision_score, recall_score,
                             f1_score)
print("accuracy is:",sum(predict_y_test==test_y)/len(test_y))
precision= precision_score(predict_y_test,test_y)
recall = recall_score(predict_y_test,test_y)
f1=f1_score(predict_y_test,test_y)
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)

%matplotlib inline
## draw chart for the cross table

from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(2)
    plt.xticks(tick_marks, [0,1])
    plt.yticks(tick_marks, [0,1])
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


# Compute confusion matrix
cm = confusion_matrix(test_y, predict_y_test)
np.set_printoptions(precision=2)
print('Confusion matrix, without normalization')
print(cm)
plt.figure()
plot_confusion_matrix(cm)
plt.show()

#--------------------
# SVM poly 2
```

```
# SVM_poly_2
#---------------------

# set the sample weights for the training model
sample_weights=[]
ratio=len(train_y)/sum(train_y==1)/10
for i in range(len(train_x)):
    if train_y[i]==0:
        sample_weights.append(1)
    else: sample_weights.append(ratio)

import time
from sklearn import svm
# training

# change the depth of the tree to 6, number of estimators=100

t=time.time()
clf = svm.SVC(C=1.0,kernel='poly',degree=2,max_iter=5000,shrinking=True, tol=0.001, verbose=False)

clf.fit(train_x_scale,train_y)

print(time.time()-t)

#testing
# test the training error
predict_y =np.array(clf.predict(train_x_scale))
print("train_accuracy is:",sum(predict_y==train_y)/len(train_y))

# test the score for the train data
from sklearn.metrics import (brier_score_loss, precision_score, recall_score,
                             f1_score)
precision= precision_score(predict_y,train_y)
recall = recall_score(predict_y,train_y)
f1=f1_score(predict_y,train_y)
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)

# define a function to prefict the result by threshold
# note: logistic model will return two probability
def predict_threshold(predict_proba, threshold):
    res=[]
    for i in range(len(predict_proba)):
        res.append(int(predict_proba[i][1]>threshold))
    return res

t=time.time()
predict_y_test=np.array(clf.predict(test_x_scale))
print("test time is:", time.time()-t)


# test the score for the train data
```

```
from sklearn.metrics import (precision_score, recall_score,
                             f1_score)
print("accuracy is:",sum(predict_y_test==test_y)/len(test_y))
precision= precision_score(predict_y_test,test_y)
recall = recall_score(predict_y_test,test_y)
f1=f1_score(predict_y_test,test_y)
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)


#draw the crosstab chart
%matplotlib inline
## draw chart for the cross table

from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(2)
    plt.xticks(tick_marks, [0,1])
    plt.yticks(tick_marks, [0,1])
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


# Compute confusion matrix
cm = confusion_matrix(test_y, predict_y_test)
np.set_printoptions(precision=2)
print('Confusion matrix, without normalization')
print(cm)
plt.figure()
plot_confusion_matrix(cm)
plt.show()
#---------------
# decision tree
#-----------------


# set the sample weights for the training model
sample_weights=[]
ratio=len(train_y)/sum(train_y==1)/10
for i in range(len(train_x)):
    if train_y[i]==0:
        sample_weights.append(1)
    else: sample_weights.append(ratio)


from sklearn import tree
# training
```

```
# change the depth of the tree to 6, number of estimators=100

t=time.time()
clf =  tree.DecisionTreeClassifier(max_depth=10,random_state= 987612345)
clf.fit(train_x_scale,train_y)

print(time.time()-t)

#testing
# test the training error
predict_y=np.array(clf.predict(train_x_scale))
print("train_accuracy is:",sum(predict_y==train_y)/len(train_y))

# test the score for the train data
from sklearn.metrics import (brier_score_loss, precision_score, recall_score,
                             f1_score)
precision= precision_score(predict_y,train_y)
recall = recall_score(predict_y,train_y)
f1=f1_score(predict_y,train_y)
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)

# define a function to prefict the result by threshold
# note: logistic model will return two probability
def predict_threshold(predict_proba, threshold):
    res=[]
    for i in range(len(predict_proba)):
        res.append(int(predict_proba[i][1]>threshold))
    return res

t=time.time()
predict_y_test_proba =np.array(clf.predict_proba(test_x_scale))
print("test time is:", time.time()-t)
predict_y_test=predict_threshold(predict_y_test_proba,0.5)


# test the score for the train data
from sklearn.metrics import (precision_score, recall_score,
                             f1_score)
print("accuracy is:",sum(predict_y_test==test_y)/len(test_y))
precision= precision_score(predict_y_test,test_y)
recall = recall_score(predict_y_test,test_y)
f1=f1_score(predict_y_test,test_y)
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)


#draw the crosstab chart
%matplotlib inline
## draw chart for the cross table
```

```
## draw chart for the cross table

from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(2)
    plt.xticks(tick_marks, [0,1])
    plt.yticks(tick_marks, [0,1])
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


# Compute confusion matrix
cm = confusion_matrix(test_y, predict_y_test)
np.set_printoptions(precision=2)
print('Confusion matrix, without normalization')
print(cm)
plt.figure()
plot_confusion_matrix(cm)
plt.show()
 #-----------------------------------------
# Adaboost
#-----------------------------------------

# set the sample weights for the training model
sample_weights=[]
ratio=len(train_y)/sum(train_y==1)/10
for i in range(len(train_x)):
    if train_y[i]==0:
        sample_weights.append(1)
    else: sample_weights.append(ratio)


from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

# training

# change the depth of the tree to 6, number of estimators=100

time_ada=time.time()
clf =  AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=10),n_estimators=100,random_state= 987612345)
clf.fit(train_x_scale,train_y)

print(time.time()-time_ada)

#testing
# test the training error
predict_y=np.array(clf.predict(train_x_scale))
print("train_accuracy is:",sum(predict_y==train_y)/len(train_y))
```

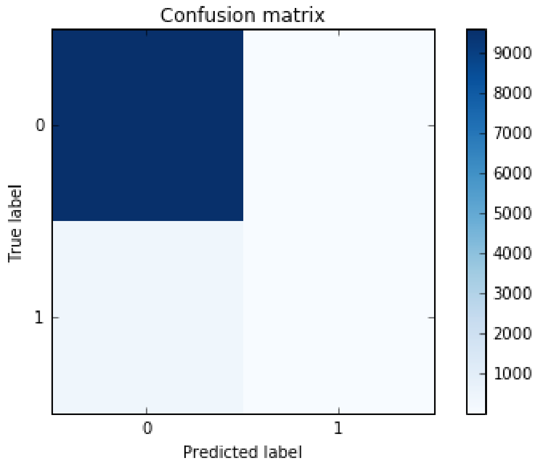
```
# test the score for the train data
from sklearn.metrics import (brier_score_loss, precision_score, recall_score,
                             f1_score)
precision= precision_score(predict_y,train_y)
recall = recall_score(predict_y,train_y)
f1=f1_score(predict_y,train_y)
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)

# define a function to prefict the result by threshold
# note: logistic model will return two probability
def predict_threshold(predict_proba, threshold):
    res=[]
    for i in range(len(predict_proba)):
        res.append(int(predict_proba[i][1]>threshold))
    return res

t=time.time()
predict_y_test_proba =np.array(clf.predict_proba(test_x_scale))
print("test time is:", time.time()-t)

predict_y_test=predict_threshold(predict_y_test_proba,0.5)


# test the score for the train data
from sklearn.metrics import (precision_score, recall_score,
                             f1_score)
print("accuracy is:",sum(predict_y_test==test_y)/len(test_y))
precision= precision_score(predict_y_test,test_y)
recall = recall_score(predict_y_test,test_y)
f1=f1_score(predict_y_test,test_y)
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)


#draw the crosstab chart
%matplotlib inline
## draw chart for the cross table

from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(2)
    plt.xticks(tick_marks, [0,1])
    plt.yticks(tick_marks, [0,1])
    plt.tight_layout()
    plt.ylabel('True label')
```

```
    plt.xlabel('Predicted label')


# Compute confusion matrix
cm = confusion_matrix(test_y, predict_y_test)
np.set_printoptions(precision=2)
print('Confusion matrix, without normalization')
print(cm)
plt.figure()
plot_confusion_matrix(cm)
plt.show()
# random forest

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

# training

# change the depth of the tree to 6, number of estimators=100

time_rf=time.time()
clf =  RandomForestClassifier(max_depth=20,n_estimators=100,random_state= 987612345)
clf.fit(train_x_scale,train_y)

print(time.time()-time_rf)

#testing
# test the training error
predict_y=np.array(clf.predict(train_x_scale))
print("train_accuracy is:",sum(predict_y==train_y)/len(train_y))

# test the score for the train data
from sklearn.metrics import (brier_score_loss, precision_score, recall_score,
                             f1_score)
precision= precision_score(predict_y,train_y)
recall = recall_score(predict_y,train_y)
f1=f1_score(predict_y,train_y)
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)

# define a function to prefict the result by threshold
# note: logistic model will return two probability
def predict_threshold(predict_proba, threshold):
    res=[]
    for i in range(len(predict_proba)):
        res.append(int(predict_proba[i][1]>threshold))
    return res

t=time.time()
predict_y_test_proba =np.array(clf.predict_proba(test_x_scale))
print("test time is:", time.time()-t)
predict_y_test=predict_threshold(predict_y_test_proba,0.5)
```

```python
# test the score for the test data
from sklearn.metrics import (precision_score, recall_score,
                             f1_score)
print("test accuracy is:",sum(predict_y_test==test_y)/len(test_y))
precision= precision_score(predict_y_test,test_y)
recall = recall_score(predict_y_test,test_y)
f1=f1_score(predict_y_test,test_y)
print("precision is: \t %s" % precision)
print("recall is: \t %s" % recall)
print("f1 score is: \t %s" %f1)
```

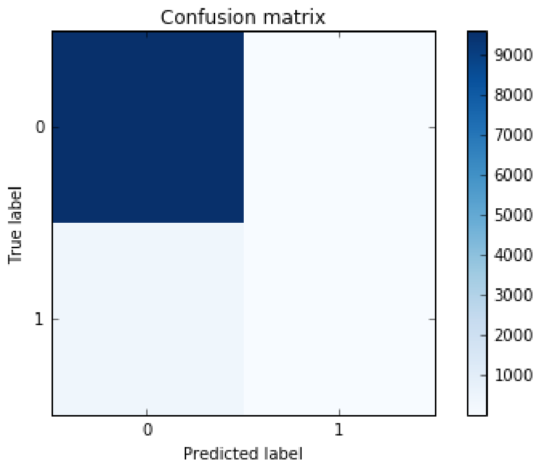
```
total array shape: (100000, 135)
train_x shape: (90000, 134)
test_x shape: (10000, 134)
test_y shape: (10000,)
train_y shape: (90000,)
[  2.99e-14   1.21e-15  -3.55e-14   1.38e-15   3.45e-14   2.58e-15
  -1.97e-14  -3.77e-15   1.91e-14  -5.28e-15  -2.02e-14   1.99e-15
   2.04e-14   9.12e-15   4.93e-14  -4.33e-15  -2.96e-14   1.20e-14
   3.12e-14   7.03e-15  -2.77e-14  -8.12e-15   7.17e-15  -3.97e-15
  -2.02e-14   2.30e-15   5.74e-14  -5.45e-15  -3.21e-14  -4.18e-15
  -4.57e-14  -3.47e-15   2.90e-14  -7.72e-15  -3.93e-15   4.46e-15
   3.77e-14   1.60e-14   3.64e-14  -5.26e-15  -8.47e-15   2.09e-15
   9.26e-15   1.21e-14  -1.35e-14   3.06e-15  -4.88e-16   1.05e-14
  -1.71e-16  -7.11e-15   4.94e-14   1.58e-14  -5.42e-15  -2.33e-14
  -5.40e-14  -2.01e-14   2.18e-14   4.32e-15   1.80e-14   3.57e-14
   7.14e-15  -2.94e-14   1.01e-14  -4.19e-14   5.29e-15   1.28e-14
  -1.03e-14  -3.41e-15  -1.59e-14  -4.38e-15   2.01e-15  -8.45e-15
   2.35e-15   1.34e-14  -3.94e-15   1.89e-14   3.56e-14  -5.85e-14
  -3.52e-14   8.80e-15   1.42e-15   7.11e-14  -9.97e-15  -3.00e-16
  -9.80e-16   1.79e-15  -1.29e-14   3.06e-15   7.99e-16   4.24e-15
   9.03e-15   3.61e-15  -1.01e-15  -1.40e-15   8.47e-16  -1.24e-14
  -6.91e-15   2.85e-15   1.08e-14  -4.62e-15  -1.32e-15   7.28e-15
   1.39e-15   6.45e-16   1.48e-15  -7.67e-16  -2.45e-15   3.87e-15
  -3.10e-15  -2.52e-15   2.03e-15  -2.39e-15  -2.52e-15  -5.88e-17
  -3.14e-15  -2.15e-15  -3.20e-16   7.96e-16   1.39e-15  -2.03e-15
   1.04e-14  -1.09e-15  -8.00e-16  -4.32e-15  -5.37e-15  -1.03e-14
   9.23e-16   1.41e-14   1.13e-14   1.52e-14  -9.82e-16   6.90e-16
  -3.23e-16   3.18e-15]
[ -2.39e-01  -4.37e-02  -2.37e-01   6.95e-02  -2.38e-01  -5.49e-02
  -2.34e-01   1.29e-01  -2.38e-01  -7.11e-02  -2.31e-01   1.23e-01
  -2.38e-01  -7.56e-02  -2.29e-01   8.45e-02  -2.39e-01  -1.20e-01
  -2.27e-01   5.09e-02  -2.40e-01  -9.53e-02  -2.26e-01   3.90e-02
  -2.40e-01  -8.71e-02  -2.25e-01   3.43e-02  -2.41e-01  -9.87e-02
  -2.22e-01   4.08e-02  -2.44e-01  -8.66e-02  -2.19e-01   2.86e-02
  -2.47e-01  -1.24e-01  -2.16e-01   9.58e-02   2.18e-02   5.62e-03
  -2.53e-02  -2.94e-02  -4.61e-02  -5.13e-02  -5.65e-02  -6.82e-02
  -8.54e-02  -1.03e-01  -2.38e-01  -2.36e-01  -2.35e-01  -2.34e-01
  -2.33e-01  -2.33e-01  -2.33e-01  -2.32e-01  -2.32e-01  -2.32e-01
   6.82e-02  -8.16e-05   4.03e-02  -2.38e-02  -9.35e-03   1.62e-03
  -1.22e-02  -4.80e-02  -6.38e-02  -8.67e-02  -7.33e-02  -4.98e-02
  -4.29e-02  -2.69e-02  -5.09e-02  -8.30e-02  -9.01e-02  -9.66e-02
```

```
  -2.41e-01  -2.27e-01  -2.34e-01   2.07e-01  -6.03e-02  -2.89e-01
  -5.50e-03  -1.13e-02  -6.87e-03  -9.90e-03  -1.88e-02  -4.03e-03
  -5.07e-03   1.07e-02   3.68e-03   1.63e-02  -7.60e-03  -7.06e-03
  -3.87e-03  -8.38e-03  -1.04e-02  -2.86e-03   3.42e-04  -1.11e-02
  -1.02e-02  -1.38e-02  -4.64e-04   3.04e-04   2.84e-03   2.30e-04
   8.03e-03  -5.75e-03   1.81e-02  -2.63e-02   6.06e-03   1.56e-02
  -7.07e-03  -1.13e-02   1.51e-02  -1.41e-02   1.89e-02   8.77e-03
  -1.66e-02   2.04e-03  -1.98e-02  -1.99e-02  -9.24e-02  -1.46e-01
  -1.05e-01  -7.59e-02  -8.31e-02  -1.34e-01  -3.04e-02  -2.01e-02
   1.46e-03  -2.87e-02]
422.86151480674744
train_accuracy is: 0.945511111111
precision is:   0.0473503929075
recall is:   0.571776155718
f1 score is:   0.0874581317454
test time is: 0.0023555755615234375
accuracy is: 0.9611
precision is:   0.0534351145038
recall is:   0.552631578947
f1 score is:   0.0974477958237
Confusion matrix, without normalization
[[9590   17]
 [ 372   21]]
```



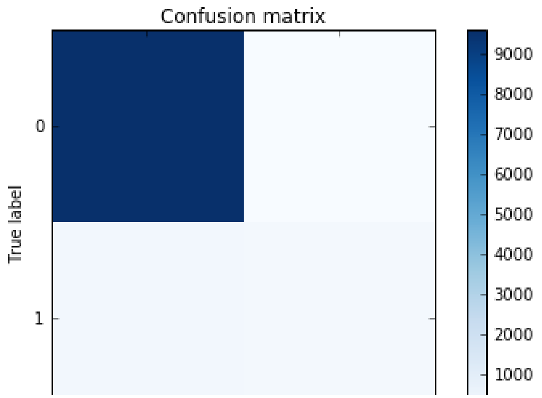

```
6.542015552520752
test time is: 0.013441801071166992
train_accuracy is: 0.945522222222
precision is:   0.0475518839412
recall is:   0.572815533981
f1 score is:   0.0878139534884
accuracy is: 0.9611
precision is:   0.0534351145038
recall is:   0.552631578947
f1 score is:   0.0974477958237
Confusion matrix, without normalization
```

```
[[9590   17]
 [ 372   21]]
```

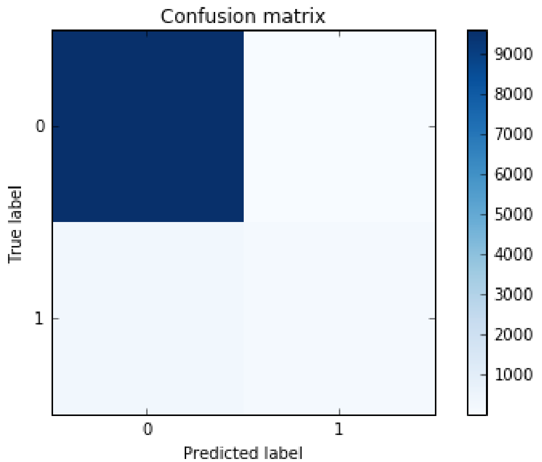



```
110.29874539375305
train_accuracy is: 0.972344444444
precision is:   0.512190207536
recall is:   0.973946360153
f1 score is:   0.671332364981
test time is: 8.987369775772095
accuracy is: 0.976
precision is:   0.399491094148
recall is:   0.975155279503
f1 score is:   0.56678700361
Confusion matrix, without normalization
[[9603    4]
 [ 236  157]]
```

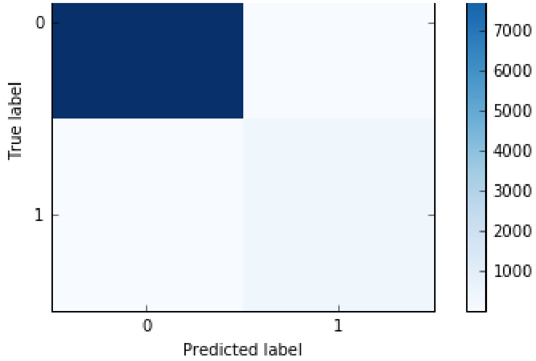
```
/home/jianwang/anaconda3/lib/python3.5/site-packages/sklearn/svm/base.py:224: ConvergenceWarning: Solver terminated early (max_iter=5000).  Consider pre-pr
ocessing your data with StandardScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
```




```
4.32948637008667
train_accuracy is: 0.962722222222
precision is:   0.337698972396
recall is:   0.961009174312
f1 score is:   0.499776353064
test time is: 0.0028090476989746094
accuracy is: 0.9721
precision is:   0.307888040712
recall is:   0.9453125
f1 score is:   0.464491362764
Confusion matrix, without normalization
[[9600    7]
 [ 272  121]]
```




```
390.4852225780487
train_accuracy is: 0.999777777778
precision is:   0.997380616563
recall is:   0.998587855558
f1 score is:   0.997983870968
test time is: 0.21262788772583008
accuracy is: 0.9964
precision is:   0.923664122137
recall is:   0.983739837398
f1 score is:   0.952755905512
Confusion matrix, without normalization
[[9601    6]
 [  30  363]]
```




```
40.48090481758118
train_accuracy is: 0.989811111111
precision is:   0.815837195245
recall is:   0.999259624877
f1 score is:   0.898280643372
test time is: 0.11994314193725586
test accuracy is: 0.9869
precision is:   0.671755725191
recall is:   0.992481203008
f1 score is:   0.801213960546
```