Limit order book

author: Jian Wang

time: 2017-06-19

Model prepare

In [ ]:

```
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

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="f
loat64"))
    data_mess_list.append(np.array(pd.read_csv(path_load+ticker_list[i]+name_mess,header=-1),dtype="flo
at64"))
    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_me
ss_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])


# 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.tx
t',\
                                                    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")


## 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
## 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)
```

Data split

In [ ]:

```
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)
```

Two classes training

Logistic regression, lasso regression, ridge regression, support vector machine, decision tree, random forest and Adaboosting are used here.

In [ ]:

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

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("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()
```

Multi-class predict

One vs One and One vs rest are used here

load the arbitrage time txt data

In [ ]:

```
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)
```

Deal with the data

In [ ]:

```
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',h
eader=-1))))

    ## print the shape of the response
## 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
```

Split the data

In [ ]:

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

ticker_ind=0
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)),:]


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))
```

One vs One

In [ ]:

```
# 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)
# 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()
```

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()
```

PnL Calculation

In [ ]:

```
def get_index(index, value):
```

```
def get_index(index, value):
    i=0
    while index[i] <value:
        i=i+1
    return i
## 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)
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()
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=[]
cost=200
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)

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()

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.savefig(ticker_list[ticker_ind]+'_cum_pnl.png')
plt.show()
```

Order book plot

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

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.
#%
#%       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"]4
   #% 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_po
s]),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()
```

Statistical properties plotting

In [ ]:

```
### 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]
```

```
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
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()
```

2) Volume plotting

In [ ]:

```
%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.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()
```

3) Intraday seasonality

In [ ]:

```
ticker_ind=0
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="lower right")
xticks=np.arange(34200,57600,2400)
plt.xticks(x[::8],xticks)
plt.show()
```

4) average shape of order books

In [ ]:

```
### 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)
```

In [ ]:

```
### 5) placement of orders
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,:]
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=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([-10000,10000])
```