1 Introduction [20 points]

Group members: Mingshu Liang, Jinrui Hou, Lin Ma
Team name: Sad Steak Sandwich
Place we got: 38
AUC score we got: 0.62251
Division of labor: each member has an equal amount of workload.

2 Overview [20 points]

Models and techniques tried: fully-connected layers, CNN, Random Forest, SVM, Log Regression, Decision Tree.

Techniques: for training fully connected network, CNN, SVM and log regression, we used SGD. For training Decision Tree, we used gradient boosting.

Out of ordinary: we add extra features by a non-linear combination of original features; we also dropped features based on the importance of the feature.

Work timeline: the first three days: we tried the most common models, such as Fully Connected Neural Network, CNN, Random Forest, logistic regression and CatBoost. The last three days: we worked on optimization and data manipulation.

3 Approach [20 points]

1. yes.

We observed data sets and found that all missing data are ”opened position qty” and ”closed position qty”. It occurred when no orders were filled in the past 500 ms. Thus all ”NaN” data should be replaced with 0. In term of adding features, we generated features by analyzing the type of data and conducting non-linear transformation. for example, we generate new data which is equal to the product of ask1 and ask1vol. this is a non-linear term and it has semantic meaning: the price ask1 agent is going to pay if the trade is closed.

we also dropped features by running the random forest. the built-in function ”feature selection.selectfrommodel” of sklearn could tell us the influence of each feature. then we dropped the features with the least influence.

RNN: the stock price is a time-indexed sequential data. however, the test data is not provided in the time order. thus we did not use this model. disadvantage: test data is not in time order

Fully Connected neural network: this is 1D sequential data with only 26 features. Thus, the fully connected neural network is a viable solution both in term of data structure and computation load even it learns features form all the input data, which means we don’t have to do feature engineering. disadvantage: computation cost is high when feature number is high.

SVM: SVM with rbf kernel could map the data into infinite high dimension and thus improve the learning capability, while at the same time keep the computing load acceptable and generalization capability at a similar level disadvantage: high computation cost when the data size is big. it could easier overfit with improper regularization setting because it can be affected noise easily. besides, it does not provide a probability output.

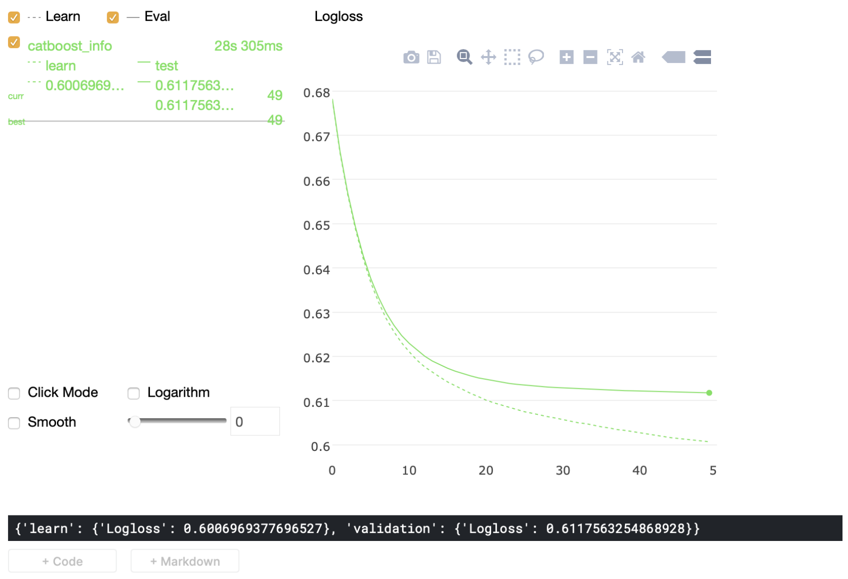
CNN: because the CNN can handle the data with high localization and shift-invariant property. it does not connect all the input data and thus save computing cost. disadvantage: not suitable for this application/input data structure as our data do not have strong localization property.

Random Forest: the predictive performance can compete with the best-supervised learning models in term of accuracy. they can also provide a reliable feature importance estimate. besides, it could be parallelized. disadvantage: high computation cost

Logistic regression: because the sample prediction has to be a probability. thus logistic regression is suitable for this application. disadvantage: cannot handle non-linearities in the data, which is a significant property of the data we have. besides, we have to carefully optimize the learning process. otherwise, the model will overfit.

4 Model Selection [20 points]

Scoring objective: Our objective is to minimize the out-of-sample loss. we tried different loss functions. For instance, cross-entropy, MSE loss and log loss. We tried to use MSE because took it as a regression problem at the beginning. Then we tried cross-entropy and log loss because we realized that we can achieve our goal by extracting the probability of the predicted classes from the classifier. Among all the models we tried, CatBoost scores the best.



```
{'learn': {'Logloss': 0.6006969377696527}, 'validation': {'Logloss': 0.6117563254868928}}
```

Figure 1: Learning curve of CatBoost (50 Iterations)

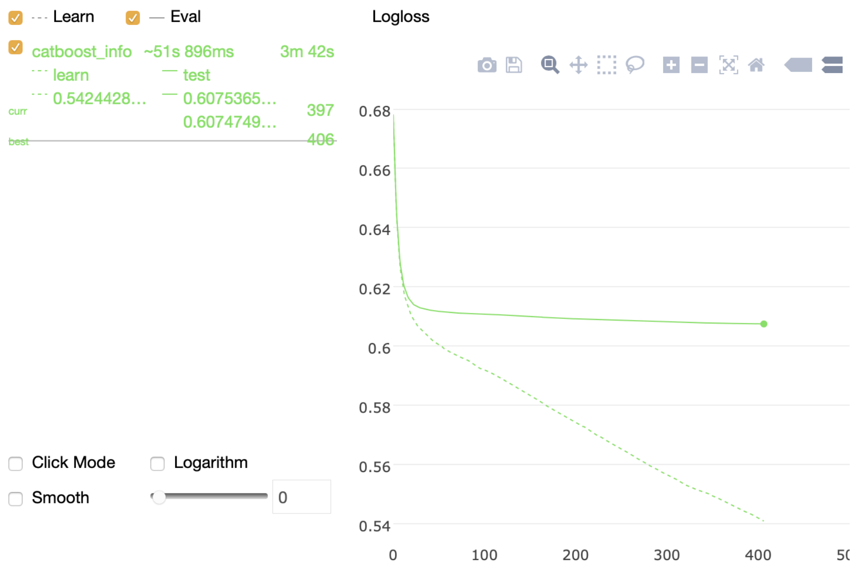


Figure 2: Learning curve of CatBoost (400 Iterations)

Validation and test: Yes, we used the validation set. We first split the given training dataset (train.csv). To be more specific, we randomly sampled 80% of the train.csv data as our training set and leave the rest 20% as our validation set.

The results of fully-connected layers, CNN both had an extremely high bias, which indicates those models didn't do effective learning and tend to give the same output no matter what input was.

SVM model gave low training error but high validation error, which indicated it was overfitting.

The results of CatBoost models showed that the model tends to become overfitting when we trained for too many iterations.

No.	Model Type	Accuracy
1	Fully connect network	64.3430%
2	CNN	64.3430%
3	Logistic Regression	65.0904%
4	SVM	64.8100%
5	Random Forest	61.2825%
6	CatBoost	0.6075

For the first 5 models, we used the percentage of accurate classification as accuracy, for CatBoost, we used logloss.

5 Conclusion [20 points]

Top features: We used the built-in function get_feature_importance() in CatBoost. It gives the following result:

Top 10 features: ask1vol, bid1vol, closed_position_qty, bid2vol, ask2vol, ask3vol, bid5vol, ask5vol, bid4vol, bid3vol.

Why AUC, advantages and disadvantages

The definition of AUC: when using normalized units, the area under the curve (often referred to as simply the AUC) is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one (assuming 'positive' ranks higher than 'negative'). Thus, by using this measure, the absolute value of the result has little effect on the final score, and only the relative ranking of the probabilities in the test set matters. This is a very loose measure. It does not care about the true distribution of the target and only cares about the ranking of the probability. A model with very high AUC score can tell us which input can give us the highest probability among several given input. but it cannot tell us what is the exact value of the probability. To get a stricter measure of the model, we may use cross-entropy.

Parallel Model:

CNN could be parallelized. First create a master model which dispatches multiple copies of itself, training in parallel on different subsets of training data. When all models finished, the parameters of different models are averaged and the master model is updated with the difference. Then the whole process is repeated until convergence.
Random forest could be parallelized. We can allocate different trees to different cores. Fully connected layers network can't be parallelized because all features are connected and calculated by iterations.

What we have learnt:

We should first do data cleaning, like filling missing data and preprocessing, like standardization.

We should select a proper model according to data properties. In our situation, data are not iid. Thus many models, like fully connected network and CNN, which are bases on the assumption that inputs are iid, didn't give a good prediction.

We can create more features by doing non-linear manipulation among existed features.

We got to know a new open-source library for gradient boosting on decision trees called CatBoost.

Challenges:

The first challenge is understanding the data. Only we firstly understand the whole process of trading can we make good manipulation of the data.

The second challenge is that the data are dependent, which makes the performance of the models not accurate enough (which we have analysed before).

stock_fully_connected

February 16, 2020

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import datasets, transforms
import pandas as pd
```

0.1 Load Data

[2]:
```
batch_size=32
all_data=pd.read_csv('./data/train.csv',dtype=np.float64)
all_data.fillna(all_data.mean(),inplace = True)
train_dataset = all_data.sample(frac=0.8, random_state=0)
test_dataset = all_data.drop(train_dataset.index)

train_tensor = torch.tensor(train_dataset.values)
test_tensor = torch.tensor(test_dataset.values)
training_data_loader=torch.utils.data.
 ↪DataLoader(train_tensor,batch_size=batch_size,shuffle=True)
test_data_loader=torch.utils.data.
 ↪DataLoader(test_tensor,batch_size=batch_size,shuffle=True)
```

[3]:
```
# look at the number of batches per epoch for training and validation
print(f'{len(training_data_loader)} training batches')
print(f'{len(training_data_loader) * batch_size} training samples')
print(f'{len(test_data_loader)} validation batches')
#print(train_tensor[0:100].numpy())
```

```
14810 training batches
473920 training samples
3703 validation batches
```

0.2 Model

[4]:
```
import torch.nn as nn

model = nn.Sequential(
    nn.Flatten(),
    nn.Linear(26, 64),
    nn.ReLU(),
    nn.Linear(64, 1)

)
```

[5]:
```
# why don't we take a look at the shape of the weights for each layer
for p in model.parameters():
    print(p.data.shape)
```

```
torch.Size([64, 26])
torch.Size([64])
torch.Size([1, 64])
torch.Size([1])
```

[6]:
```
 # our model has some # of parameters:
count = 0
for p in model.parameters():
    n_params = np.prod(list(p.data.shape)).item()
    count += n_params
print(f'total params: {count}')
```

```
total params: 1793
```

[7]:
```
import torch.optim as optim
loss_fn = nn.MSELoss()
#optimizer = optim.Adam(model.parameters())
optimizer = optimizer = optim.SGD(model.parameters(), lr=0.001)
```

[8]:
```
# Some layers, such as Dropout, behave differently during training
model.train()

for epoch in range(5):
    for i, features in enumerate(training_data_loader):
        data=features[:,1:-1]
        target=features[:,-1]
        target=target.resize(len(target),1)
        #print(data, target)

        # Erase accumulated gradients
        optimizer.zero_grad()
        # Forward pass
        output = model(data.float())

        # Calculate loss
        loss = loss_fn(output, target.float())

        # Backward pass
        loss.backward()

        # Weight update
        optimizer.step()

    # Track loss each epoch
    print('Train Epoch: %d  Loss: %.4f' % (epoch + 1,  loss.item()))
```

```
/opt/anaconda3/lib/python3.7/site-packages/torch/tensor.py:362: UserWarning:
non-inplace resize is deprecated
  warnings.warn("non-inplace resize is deprecated")
```

```
Train Epoch: 1  Loss: 0.1808
Train Epoch: 2  Loss: 0.2347
Train Epoch: 3  Loss: 0.1820
Train Epoch: 4  Loss: 0.2536
Train Epoch: 5  Loss: 0.1987
```

[9]:
```
# Putting layers like Dropout into evaluation mode
model.eval()

test_loss = 0
correct = 0

# Turning off automatic differentiation
with torch.no_grad():
    for features in test_data_loader:
        data=features[:,1:-1]
        target=features[:,-1]
        target=target.resize(len(target),1)

        output = model(data.float())
        test_loss += loss_fn(output, target.float()).item()  # Sum up batch loss
        pred = output.argmax(dim=1, keepdim=True)  # Get the index of the max␣
 ↪class score
        correct += pred.eq(target.view_as(pred)).sum().item()

test_loss /= len(test_data_loader.dataset)

print('Test set: Average loss: %.4f, Accuracy: %d/%d (%.4f)' %
      (test_loss, correct, len(test_data_loader.dataset),
       100. * correct / len(test_data_loader.dataset)))
```

```
Test set: Average loss: 0.0072, Accuracy: 76231/118476 (64.3430)
```

[ ]:

```
```

[ ]:

```
```

[ ]:

```
```

stock_CNN

February 16, 2020

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import datasets, transforms
import pandas as pd

from sklearn.preprocessing import StandardScaler
```

0.1 Load Data

[3]:
```
batch_size=32
all_data=pd.read_csv('./train.csv',dtype=np.float64)
all_data.fillna(all_data.mean(),inplace = True)

all_data[['last_price','mid','opened_position_qty ','closed_position_qty',\
          'transacted_qty','d_open_interest','bid1','bid2','bid3','bid4',\
          'bid5','ask1','ask2','ask3','ask4','ask5','bid1vol','bid2vol',\
         ␣
 ↪'bid3vol','bid4vol','bid5vol','ask1vol','ask2vol','ask3vol','ask4vol','ask5vol']]\
          = StandardScaler().
 ↪fit_transform(all_data[['last_price','mid','opened_position_qty␣
 ↪','closed_position_qty',\
          'transacted_qty','d_open_interest','bid1','bid2','bid3','bid4',\
          'bid5','ask1','ask2','ask3','ask4','ask5','bid1vol','bid2vol',\
         ␣
 ↪'bid3vol','bid4vol','bid5vol','ask1vol','ask2vol','ask3vol','ask4vol','ask5vol']])

train_dataset = all_data.sample(frac=0.8, random_state=0)
test_dataset = all_data.drop(train_dataset.index)

train_tensor = torch.tensor(train_dataset.values)
test_tensor = torch.tensor(test_dataset.values)
```

```
training_data_loader=torch.utils.data.
 ↪DataLoader(train_tensor,batch_size=batch_size,shuffle=True)
test_data_loader=torch.utils.data.
 ↪DataLoader(test_tensor,batch_size=batch_size,shuffle=True)
```

[4]:
```
# look at the number of batches per epoch for training and validation
print(f'{len(training_data_loader)} training batches')
print(f'{len(training_data_loader) * batch_size} training samples')
print(f'{len(test_data_loader)} validation batches')
#print(train_tensor[0:100].numpy())
```

```
14810 training batches
473920 training samples
3703 validation batches
```

0.2 Model

[5]:
```
import torch.nn as nn

model = nn.Sequential(
    nn.Conv1d(in_channels=1, out_channels=128, kernel_size=5, stride=1),
    nn.BatchNorm1d(128),
    nn.ReLU(),
    nn.Dropout(p=0.2),

    nn.Conv1d(in_channels=128, out_channels=64, kernel_size=5, stride=1),
    nn.BatchNorm1d(64),
    nn.ReLU(),
    nn.Dropout(p=0.2),

    nn.Conv1d(in_channels=64, out_channels=64, kernel_size=5, stride=1),
    nn.BatchNorm1d(64),
    nn.ReLU(),
    nn.Dropout(p=0.2),

    nn.Conv1d(in_channels=64, out_channels=32, kernel_size=3, stride=1),
    nn.BatchNorm1d(32),
    nn.ReLU(),
    nn.MaxPool1d(2),
    nn.Dropout(p=0.2),

    nn.Flatten(),
    nn.Linear(192, 64),
    nn.ReLU(),
    nn.Linear(64, 2),

)
```

[6]:
```
# why don't we take a look at the shape of the weights for each layer
for p in model.parameters():
    print(p.data.shape)
```

```
torch.Size([128, 1, 5])
torch.Size([128])
torch.Size([128])
torch.Size([128])
torch.Size([64, 128, 5])
torch.Size([64])
torch.Size([64])
torch.Size([64])
torch.Size([64, 64, 5])
torch.Size([64])
torch.Size([64])
torch.Size([64])
torch.Size([32, 64, 3])
torch.Size([32])
torch.Size([32])
torch.Size([32])
torch.Size([64, 192])
torch.Size([64])
torch.Size([1, 64])
torch.Size([1])
```

[7]:
```
 # our model has some # of parameters:
count = 0
for p in model.parameters():
    n_params = np.prod(list(p.data.shape)).item()
    count += n_params
print(f'total params: {count}')
```

```
total params: 81505
```

[8]:
```
import torch.optim as optim

#myweight = torch.from_numpy(np.array([0.7, 0.3])).type('torch.FloatTensor')

criterion = nn.MSELoss()
optimizer = optim.RMSprop(model.parameters())
```

[9]:
```
# Some layers, such as Dropout, behave differently during training
model.train()

for epoch in range(5):
    for i, features in enumerate(training_data_loader):
        data_temp = features[:,1:-1]
        data = torch.from_numpy(np.expand_dims(data_temp, axis=1)).type('torch.
↪FloatTensor')
        #print(data.shape)

        target = np.transpose(features[:,-1])
        #print(target.shape)
        #target = target.reshape(1,1,batch_size)
        #print(data, target)

        # Erase accumulated gradients
        optimizer.zero_grad()

        # Forward pass
        output = model(data)
        #print(output.shape)

        # Calculate loss
        loss = criterion(output, target.float())

        # Backward pass
        loss.backward()

        # Weight update
        optimizer.step()

    # Track loss each epoch
    print('Train Epoch: %d  Loss: %.4f' % (epoch + 1,  loss.item()))
```

```
/opt/miniconda3/lib/python3.7/site-packages/torch/nn/modules/loss.py:431:
UserWarning: Using a target size (torch.Size([32])) that is different to the
input size (torch.Size([32, 1])). This will likely lead to incorrect results due
to broadcasting. Please ensure they have the same size.
  return F.mse_loss(input, target, reduction=self.reduction)
/opt/miniconda3/lib/python3.7/site-packages/torch/nn/modules/loss.py:431:
UserWarning: Using a target size (torch.Size([16])) that is different to the
input size (torch.Size([16, 1])). This will likely lead to incorrect results due
to broadcasting. Please ensure they have the same size.
  return F.mse_loss(input, target, reduction=self.reduction)
```

```
Train Epoch: 1  Loss: 0.2152
Train Epoch: 2  Loss: 0.2479
Train Epoch: 3  Loss: 0.2487
Train Epoch: 4  Loss: 0.2898
Train Epoch: 5  Loss: 0.1927
```

```
[10]: # Putting layers like Dropout into evaluation mode
      model.eval()

      test_loss = 0
      correct = 0

      # Turning off automatic differentiation
      with torch.no_grad():
          for features in test_data_loader:
              data_temp = features[:,1:-1]
              data = torch.from_numpy(np.expand_dims(data_temp, axis=1)).type('torch.
       ↪FloatTensor')
              #print(data.shape)

              target = np.transpose(features[:,-1])

              output = model(data)
              #print(output)
              test_loss += criterion(output, target.long()).item()  # Sum up batch loss
              pred = output.argmax(dim=1, keepdim=True)  # Get the index of the max␣
       ↪class score
              correct += pred.eq(target.view_as(pred)).sum().item()

      test_loss /= len(test_data_loader.dataset)

      print('Test set: Average loss: %.4f, Accuracy: %d/%d (%.4f)' %
            (test_loss, correct, len(test_data_loader.dataset),
             100. * correct / len(test_data_loader.dataset)))
```

```
Test set: Average loss: 0.0072, Accuracy: 76231/118476 (64.3430)
```

```
/opt/miniconda3/lib/python3.7/site-packages/torch/nn/modules/loss.py:431:
UserWarning: Using a target size (torch.Size([12])) that is different to the
input size (torch.Size([12, 1])). This will likely lead to incorrect results due
to broadcasting. Please ensure they have the same size.
  return F.mse_loss(input, target, reduction=self.reduction)
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

logistic regression

February 16, 2020

1 Log Regression

[1]:
```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler

import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import datasets, transforms

%matplotlib inline
import pandas as pd
import csv
```

1.0.1 Load the data:

[2]:
```
batch_size=32
all_data_raw=pd.read_csv('./caltech-cs155-2020/train.csv',dtype=np.float64)

##replace NaN with column mean
all_data_raw.fillna(all_data_raw.mean(),inplace = True)

all_data = all_data_raw.drop(columns=['id','opened_position_qty␣
 ↪','closed_position_qty',\
          'transacted_qty','d_open_interest','bid1','bid2','bid3','bid4',\
          'bid5','ask1','ask2','ask3','ask4','ask5'])

all_data[['last_price','mid','bid1vol','bid2vol',\
         ␣
 ↪'bid3vol','bid4vol','bid5vol','ask1vol','ask2vol','ask3vol','ask4vol','ask5vol']]\
= StandardScaler().
 ↪fit_transform(all_data[['last_price','mid','bid1vol','bid2vol',\
        ␣
↪'bid3vol','bid4vol','bid5vol','ask1vol','ask2vol','ask3vol','ask4vol','ask5vol']])

train_dataset = all_data.sample(frac=0.8, random_state=0)
test_dataset = all_data.drop(train_dataset.index)

Train = train_dataset.to_numpy()
Test = test_dataset.to_numpy()

X = Train[:,0:-1]
y = Train[:,-1]

X_test = Test[:,0:-1]
Y_test = Test[:,-1]

print(X)
```

```
[[ 1.02883736  1.02762608 -0.42908005 ... -0.72780031 -0.55509068
  -0.56309797]
 [-0.75352544 -0.74985659  0.97626763 ... -0.72780031  0.98199513
  -0.41973541]
 [-0.97601517 -0.9747925  -0.66330467 ... -0.72780031 -0.09396494
  -0.70646053]
 ...
 [-1.63614953 -1.63737545  0.03936917 ... -0.55890869  0.21345223
  -0.56309797]
 [ 0.68654546  0.68777726 -0.66330467 ... -0.39001708 -0.09396494
  -0.70646053]
 [ 0.22934129  0.22323571  0.74204301 ...  0.28554936  1.59682946
   1.87406555]]
```

2 Conduct Log Regression

[3]:
```
#==============================================
# this is log regression

# perform the log regression
clf = LogisticRegression(solver='lbfgs', C=1000).fit(X, y)
# output the regression accuracy
print('the average regression accuracy for logistic regression is ',clf.score(X,␣
↪y))

#plot decision boundary
#make_plot(X, y, clf, 'log regression', 'log regression')

#==============================================
the average regression accuracy for logistic regression is  0.6509039805530233
```

[4]:
```
print(clf.score(X_test, Y_test))
print(Y_test)
```

```
0.6517100509807894
[1. 1. 0. ... 1. 0. 0.]
```

[ ]:
```
# this is
```

[5]:
```
result = clf.predict_proba(X_test)
print(result[:,-1].T)
```

```
[0.57531644 0.53634684 0.13604255 ... 0.77247662 0.12385601 0.12385601]
```

[19]:
```
all_data_test=pd.read_csv('./caltech-cs155-2020/test.csv',dtype=np.float64)

##replace NaN with column mean
all_data_test.fillna(all_data_test.mean(),inplace = True)

all_data = all_data_test.drop(columns=['id','opened_position_qty␣
 ↪','closed_position_qty',\
           'transacted_qty','d_open_interest','bid1','bid2','bid3','bid4',\
           'bid5','ask1','ask2','ask3','ask4','ask5'])

all_data[['last_price','mid','bid1vol','bid2vol',\
         ␣
 ↪'bid3vol','bid4vol','bid5vol','ask1vol','ask2vol','ask3vol','ask4vol','ask5vol']]\
= StandardScaler().
 ↪fit_transform(all_data[['last_price','mid','bid1vol','bid2vol',\
         ␣
 ↪'bid3vol','bid4vol','bid5vol','ask1vol','ask2vol','ask3vol','ask4vol','ask5vol']])

test_dataset = all_data.sample(frac=1, random_state=0)

test = test_dataset.to_numpy()

X = test[:,0:-1]
print(test.shape)

id=all_data_test["id"]
id_numpy=id.to_numpy()
id_numpy=id_numpy[1:473905]
print(id_numpy.shape)
```

```
(191859, 12)
(191858,)
```

[23]:
```
result = clf.predict_proba(test)
result_1 = result[0:191858,-1]
print(result_1)
print(result_1.shape)
```

```
[0.30628002 0.15856852 0.45739955 ... 0.7418655  0.65310803 0.19953139]
(191858,)
```

[24]:
```
df = pd.DataFrame({"id" : id_numpy, "Predicted" : result_1})
#df['id'] = df['id'].astype(int)
df.to_csv("submission2.csv", index=False)
```

[ ]:
```

```

stock_svm

February 16, 2020

[1]:
```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVR
import numpy as np
import csv
from sklearn.ensemble import RandomForestRegressor
import scipy
from scipy.stats.stats import pearsonr
from scipy import stats
from matplotlib import pyplot as plt
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn import metrics
```

[2]:
```
all_data=pd.read_csv('./data/train.csv')
##replace NaN with column mean
all_data.fillna(all_data.mean(),inplace = True)

#df = all_data.drop(["id","opened_position_qty " ,"closed_position_qty"␣
 ↪,"transacted_qty","d_open_interest","bid1","bid2","bid3","bid4","bid5","ask1","ask2","ask3",'
all_data["bid1sum"]=all_data["bid1"]*all_data["bid1vol"]
all_data["bid2sum"]=all_data["bid2"]*all_data["bid2vol"]
all_data["bid3sum"]=all_data["bid3"]*all_data["bid3vol"]
all_data["bid4sum"]=all_data["bid4"]*all_data["bid4vol"]
all_data["bid5sum"]=all_data["bid5"]*all_data["bid5vol"]
all_data["ask1sum"]=all_data["ask1"]*all_data["ask1vol"]
all_data["ask2sum"]=all_data["ask2"]*all_data["ask2vol"]
all_data["ask3sum"]=all_data["ask3"]*all_data["ask3vol"]
all_data["ask4sum"]=all_data["ask4"]*all_data["ask4vol"]
all_data["ask5sum"]=all_data["ask5"]*all_data["ask5vol"]



##downsizing for test
all_data=all_data.sample(frac=0.01, random_state=1)

train_dataset = all_data.sample(frac=0.8, random_state=1)
test_dataset = all_data.drop(train_dataset.index)
```

```
Train = train_dataset.to_numpy()
Test = test_dataset.to_numpy()

X_train = Train[:,1:-1]
Y_train = Train[:,-1]
print('y=1', np.sum(Y_train, axis=0))

X_test = Test[:,1:-1]
Y_test = Test[:,-1]


# Applying feature scaling on the train data
min_max_scaler = MinMaxScaler()
X_test = min_max_scaler.fit_transform(X_test)
X_train = min_max_scaler.fit_transform(X_train)
print('X test',X_test[1])

C_vals=[1,10,1e2,1e3,1e4,1e5]
gamma_vals=[1e-12, 1e-9, 1e-6, 1e-3, 1,1e2]
for i in range(1):
    for j in range(1):
        #C_val=C_vals[i]
        C_val=1
        #gamma_val=gamma_vals[j]
        gamma_val=0.001
        print('=============================')
        print(C_val,gamma_val)

        classifier = SVR(kernel = 'rbf', C=C_val, gamma=gamma_val)

        # Fitting Linear Kernel SVM to the Training set
        # classifier = SVC(kernel = 'linear', random_state = 0)

        classifier.fit(X_train, Y_train)
        prediction = classifier.predict(X_test)
        print(prediction)
        loss = np.sum(abs(prediction-Y_test),axis = 0)

        print(1-loss/len(Y_test))


        vali_data=pd.read_csv('./data/test.csv')
        vali_data.fillna(all_data.mean(),inplace = True)
        #vali_data=vali_data.drop(["id","opened_position_qty "␣
 ↪,"closed_position_qty"␣
 ↪,"transacted_qty","d_open_interest","bid1","bid2","bid3","bid4","bid5","ask1","ask2","ask3",'
        vali_data["bid1sum"]=vali_data["bid1"]*vali_data["bid1vol"]
        vali_data["bid2sum"]=vali_data["bid2"]*vali_data["bid2vol"]
        vali_data["bid3sum"]=vali_data["bid3"]*vali_data["bid3vol"]
        vali_data["bid4sum"]=vali_data["bid4"]*vali_data["bid4vol"]
        vali_data["bid5sum"]=vali_data["bid5"]*vali_data["bid5vol"]
        vali_data["ask1sum"]=vali_data["ask1"]*vali_data["ask1vol"]
        vali_data["ask2sum"]=vali_data["ask2"]*vali_data["ask2vol"]
        vali_data["ask3sum"]=vali_data["ask3"]*vali_data["ask3vol"]
        vali_data["ask4sum"]=vali_data["ask4"]*vali_data["ask4vol"]
        vali_data["ask5sum"]=vali_data["ask5"]*vali_data["ask5vol"]

        vali_data=vali_data.to_numpy()
        vali_data_id=vali_data[:,0]
        vali_data=vali_data[:,1:]

        vali_predict=classifier.predict(vali_data)
        print(vali_predict[1:30])

        vali_df = pd.DataFrame({"id" : vali_data_id, "Predicted" : vali_predict})
        vali_df['id'] = vali_df['id'].astype(int)
        vali_df.to_csv("submission2.csv", index=False)
```

```
y=1 112467722.6
X test [0.71322034 0.71225457 0.06669445 0.0892853  0.         0.58333333
 0.71225457 0.71225457 0.71370421 0.71370421 0.71370421 0.71225457
 0.71205962 0.71186441 0.71254237 0.71254237 0.04878049 0.1025641
 0.01886792 0.         0.02739726 0.         0.05263158 0.
 0.02020202 0.01204819 0.         0.04964193 0.10894998 0.02088447
 0.00145456 0.02917758 0.00158062 0.05568534 0.00099343 0.0215059 ]
==============================
1 0.001
[15624.9601327  15625.16665253 15625.02835104 ... 15624.99392718
 15625.21698274 15625.66042707]
-15403.74110426831
[15625.09834639 15625.09834639 15625.09834639 15625.09834639
 15625.09834639 15625.09834639 15625.09834639 15625.09834639
 15625.09834639 15625.09834639 15625.09834639 15625.09834639
 15625.09834639 15625.09834639 15625.09834639 15625.09834639
 15625.09834639 15625.09834639 15625.09834639 15625.09834639
 15625.09834639 15625.09834639 15625.09834639 15625.09834639
 15625.09834639 15625.09834639 15625.09834639 15625.09834639
 15625.09834639]
```

[ ]:

[ ]:

```
[ ]:
```

Random_forest_preprocess

February 16, 2020

[13]:
```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
import numpy as np
from sklearn.model_selection import train_test_split
```

[14]:
```
all_data=pd.read_csv('./data/train.csv',dtype=np.float64)

##replace NaN with column mean
all_data.fillna(all_data.mean(),inplace = True)

train_dataset = all_data.sample(frac=0.8, random_state=0)
test_dataset = all_data.drop(train_dataset.index)

Train = train_dataset.to_numpy()
Test = test_dataset.to_numpy()

X_train = Train[:,1:-1]
Y_train = Train[:,-1]

X_test = Test[:,1:-1]
Y_test = Test[:,-1]
```

[15]:
```
sel = SelectFromModel(RandomForestClassifier(n_estimators = 100))
sel.fit(X_train, Y_train)
```

[15]:
```
SelectFromModel(estimator=RandomForestClassifier(bootstrap=True,
                                                 class_weight=None,
                                                 criterion='gini',
                                                 max_depth=None,
                                                 max_features='auto',
                                                 max_leaf_nodes=None,
                                                 min_impurity_decrease=0.0,
                                                 min_impurity_split=None,
                                                 min_samples_leaf=1,
                                                 min_samples_split=2,
                                                 min_weight_fraction_leaf=0.0,
                                                 n_estimators=100, n_jobs=None,
                                                 oob_score=False,
                                                 random_state=None, verbose=0,
                                                 warm_start=False),
                max_features=None, norm_order=1, prefit=False, threshold=None)
```

[16]:
```
sel.get_support()
```

[16]:
```
array([ True,  True, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True])
```

[ ]:

[ ]:

[ ]:

[ ]:

stock_randomforest

February 16, 2020

[6]:
```
import numpy as np
import csv
import random
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
import scipy
from scipy.stats.stats import pearsonr
from scipy import stats
from matplotlib import pyplot as plt
import pandas as pd
```

[7]:
```
batch_size=32
all_data_raw=pd.read_csv('./train.csv',dtype=np.float64)

##replace NaN with column mean
all_data_raw.fillna(all_data_raw.mean(),inplace = True)

all_data = all_data_raw.drop(columns=['id','opened_position_qty␣
 ↪','closed_position_qty',\
          'transacted_qty','d_open_interest','bid1','bid2','bid3','bid4',\
          'bid5','ask1','ask2','ask3','ask4','ask5'])

all_data[['last_price','mid','bid1vol','bid2vol',\
         ␣
 ↪'bid3vol','bid4vol','bid5vol','ask1vol','ask2vol','ask3vol','ask4vol','ask5vol']]\
= StandardScaler().
 ↪fit_transform(all_data[['last_price','mid','bid1vol','bid2vol',\
         ␣
 ↪'bid3vol','bid4vol','bid5vol','ask1vol','ask2vol','ask3vol','ask4vol','ask5vol']])

frac = 0.8

train_dataset = all_data.head(int(len(all_data)*frac))
test_dataset = all_data.drop(train_dataset.index)

Train = train_dataset.to_numpy()
```

```
Test = test_dataset.to_numpy()
```

[8]:
```
X_train = Train[:,0:-1]
Y_train = Train[:,-1]

X_test = Test[:,0:-1]
Y_test = Test[:,-1]

print(X_test.shape)
```

```
(118476, 12)
```

[9]:
```
clf = RandomForestClassifier(n_estimators=100, criterion='gini', max_depth=None)
clf.fit(X_train, Y_train)
```

[9]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

[10]:
```
loss = 0;
correct = 0;

#for i in range(len(Y_test)):

Y_out = clf.predict(X_test)

#Y_label = np.where(Y_out < 0.5, 0, 1)

loss = np.sum(abs(Y_out-Y_test),axis = 0)
    #if Y_out > 0.5:
    #    Y_pred = 1
    #else:
    #    Y_pred = 0

    #if Y_pred == Y_test[i]:
    #    correct += 1

    #loss += Y_test - Y_out
print(Y_out)
print(loss)
print(1-loss/len(Y_test))
```

```
[0. 1. 0. ... 0. 0. 0.]
```

```
45871.0
0.6128245383031162
```

[ ]:

```

```

stock_CatBoost

February 18, 2020

- The temporal order is scrambled in the test data making TS useless there.
- Still, just for learning / realism, we can still do it in the training data!
- Let's add pseudo dates, and aggregate features on column subsets. Finally i'll run a model to predict the target!

[ ]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

from datetime import datetime
from scipy.special import logsumexp

from catboost import Pool, cv, CatBoostClassifier, CatBoostRegressor
from sklearn.metrics import mean_squared_error, classification_report
```

[ ]:
```
train = pd.read_csv("/kaggle/input/caltech-cs155-2020/train.csv")
test = pd.read_csv("/kaggle/input/caltech-cs155-2020/test.csv")
df = pd.concat([train,test],sort=False)
print(df.shape)
print(df.columns)
df.tail()
```

[ ]:
```
train.head()
```

[ ]:
```
test.head()
```

[ ]:
```
## y is binary.
display(train["y"].describe())
```

[ ]:
```
bid_cols = ['bid1','bid2', 'bid3', 'bid4', 'bid5']
bid_vol_cols = ['bid1vol', 'bid2vol', 'bid3vol', 'bid4vol', 'bid5vol']
ask_cols = ['ask1', 'ask2', 'ask3', 'ask4', 'ask5',]
ask_vol_cols = ['ask1vol','ask2vol', 'ask3vol', 'ask4vol', 'ask5vol']
```

```
group_cols = {"bid_cols":bid_cols,"bid_vol_cols":bid_vol_cols,"ask_cols":
↪ask_cols,"ask_vol_cols":ask_vol_cols}
```

- Additional features could include: rank, which bid number is the max/min, etc'
- features between the aggregated features (e.g. max bid div max ask..)

```
[ ]: for group in group_cols.keys():
         print(group)
         df[f"{group}_max"] = df[group_cols[group]].max(axis=1)
         df[f"{group}_min"] = df[group_cols[group]].min(axis=1)
         df[f"{group}_spread"] = df[f"{group}_max"].div(df[f"{group}_min"])
         df[f"{group}_logsumexp"] = df[group_cols[group]].apply(logsumexp)

         df[f"{group}_max"] = df[group_cols[group]].max(axis=1)

     df["last_price_div__mid"] = df["last_price"].div(df["mid"])
```

Additional features

```
[ ]: df["bid1sum"]=df["bid1"]*df["bid1vol"]
     df["bid2sum"]=df["bid2"]*df["bid2vol"]
     df["bid3sum"]=df["bid3"]*df["bid3vol"]
     df["bid4sum"]=df["bid4"]*df["bid4vol"]
     df["bid5sum"]=df["bid5"]*df["bid5vol"]
     df["ask1sum"]=df["ask1"]*df["ask1vol"]
     df["ask2sum"]=df["ask2"]*df["ask2vol"]
     df["ask3sum"]=df["ask3"]*df["ask3vol"]
     df["ask4sum"]=df["ask4"]*df["ask4vol"]
     df["ask5sum"]=df["ask5"]*df["ask5vol"]
     df["extra1"]=df["opened_position_qty "]*df["closed_position_qty"]
     df["extra2"]=df["opened_position_qty "]*df["transacted_qty"]
     df["extra3"]=df["transacted_qty"]*df["closed_position_qty"]
     df["extra4"]=df["d_open_interest"]*df["opened_position_qty "]
     df["extra5"]=df["d_open_interest"]*df["transacted_qty"]
     df["extra6"]=df["d_open_interest"]*df["closed_position_qty"]
     '''
     df["bid1sq"]=df["bid1"]*df["bid1"]
     df["bid2sq"]=df["bid2"]*df["bid2"]
     df["bid3sq"]=df["bid3"]*df["bid3"]
     df["bid4sq"]=df["bid4"]*df["bid4"]
     df["bid5sq"]=df["bid5"]*df["bid5"]
     df["ask1sq"]=df["ask1"]*df["ask1"]
     df["ask2sq"]=df["ask2"]*df["ask2"]
     df["ask3sq"]=df["ask3"]*df["ask3"]
     df["ask4sq"]=df["ask4"]*df["ask4"]
     df["ask5sq"]=df["ask5"]*df["ask5"]
     df["bid1volsq"]=df["bid1vol"]*df["bid1vol"]
     df["bid2volsq"]=df["bid2vol"]*df["bid2vol"]
```

```
df["bid3volsq"]=df["bid3vol"]*df["bid3vol"]
df["bid4volsq"]=df["bid4vol"]*df["bid4vol"]
df["bid5volsq"]=df["bid5vol"]*df["bid5vol"]
df["ask1volsq"]=df["ask1vol"]*df["ask1vol"]
df["ask2volsq"]=df["ask2vol"]*df["ask2vol"]
df["ask3volsq"]=df["ask3vol"]*df["ask3vol"]
df["ask4volsq"]=df["ask4vol"]*df["ask4vol"]
df["ask5volsq"]=df["ask5vol"]*df["ask5vol"]
'''

df=df.
 ↪drop(["bid_vol_cols_min","ask2","bid1","ask4","bid5","ask3","bid_cols_min","last_price"],axis
```

```
[ ]: df["date"] = pd.to_datetime("1.1.2019")
     df["date"] = df["date"] + pd.to_timedelta(df["id"]/2,unit="s") # 500 ms per row

     df["date"].describe()
```

1 Split back into train and test, and build model

```
[ ]: all_train = df.loc[~df.y.isna()]
     train = all_train.sample(frac=0.7, random_state=1)
     vali = all_train.drop(train.index)
     print(f"train shape {train.shape[0]}")
     print(f"vali shape {vali.shape[0]}")
     test = df.loc[df.y.isna()]
     print(f"test shape {test.shape[0]}")
```

```
[ ]: train.drop(["id"],axis=1).to_csv("train_hft.csv.
      ↪gz",index=False,compression="gzip")
     vali.drop(["id"],axis=1).to_csv("vali_hft.csv.gz",index=False,compression="gzip")
     test.to_csv("test_hft_nodates.csv.gz",index=False,compression="gzip")
```

```
[ ]: # we don't know if the test set has a temporal split, so we'll just try a random␣
      ↪split for now
     X = train.drop(["id","date","y"],axis=1)
     y = train["y"]
     X_vali = vali.drop(["id","date","y"],axis=1)
     y_vali = vali["y"]
```

```
[ ]: train_pool = Pool(data=X,label = y)
     vali_pool = Pool(data=X_vali,label = y_vali)
```

```
[ ]: '''
     ### hyperparameter tuning example grid for catboost :
```

```
grid = {'learning_rate': [0.05, 0.1],
        'depth': [6, 11],
         'l2_leaf_reg': [1, 3,9],
        "iterations": [30],
        "custom_metric":['Logloss', 'AUC']}

model = CatBoostClassifier()

## can also do randomized search - more efficient typically, especially for␣
 ↪large search space - `randomized_search`
grid_search_result = model.grid_search(grid,
                                       train_pool,
                                       plot=True,
                                      refit = True, #  refit best model on all␣
 ↪data
                                      partition_random_seed=42)

print(model.get_best_score())
'''
```

[ ]:
```
model = CatBoostClassifier(learning_rate=0.
 ↪1,depth=11,l2_leaf_reg=9,iterations=500)

#model.fit(train_pool, plot=True,silent=True)
model.fit(X=X,y=y, eval_set=vali_pool, plot=True,silent=True)
print(model.get_best_score())
```

1.1 Features importances

[ ]:
```
feature_importances = model.get_feature_importance(train_pool)
feature_names = X.columns
for score, name in sorted(zip(feature_importances, feature_names), reverse=True):
    if score > 0.2:
        print('{0}: {1:.2f}'.format(name, score))
```

[ ]:
```
import shap
#shap.initjs()

#explainer = shap.TreeExplainer(model)
#shap_values = explainer.shap_values(train_pool)

#visualize the training set predictions
#SHAP plots for all the data is very slow, so we'll only do it for a sample.␣
 ↪Taking the head instead of a random sample is dangerous!
#shap.force_plot(explainer.expected_value,shap_values[0,:300], X.iloc[0,:300])
```

[ ]:
```
# summarize the effects of all the features
#shap.summary_plot(shap_values, X)
```

[ ]:
```
## todo : PDP features +- from shap
```

1.2 export predictions

[ ]:
```
test["Predicted"] = model.predict(test.
 ↪drop(["id","date","y",],axis=1),prediction_type='Probability')[:,1]
test[["id","Predicted"]].to_csv("submission6.csv",index=False)
```