Report of project 4

Tongzhe Zhang 903181877

1.

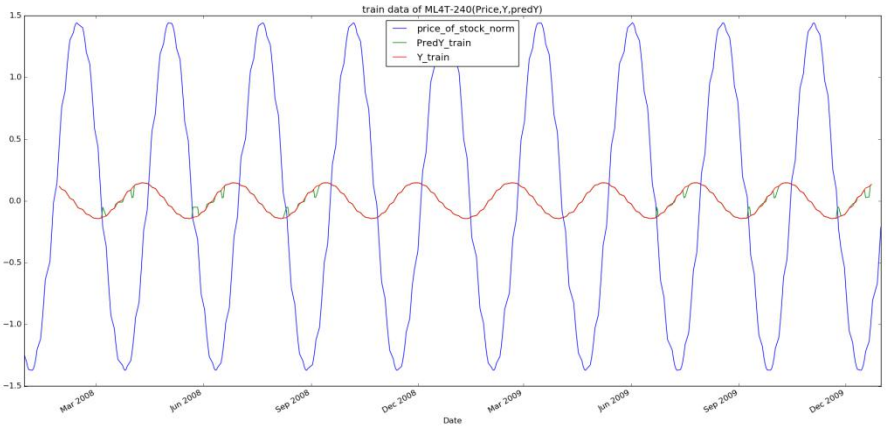

Chart 1 Sine data in-sample Training Y/Price/Predicted Y

2.

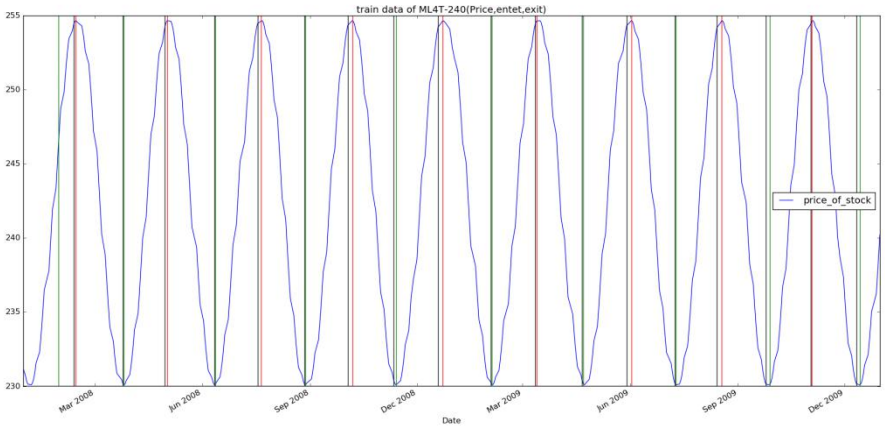

Chart 2 Sine data in-sample Entries/Exits

3.

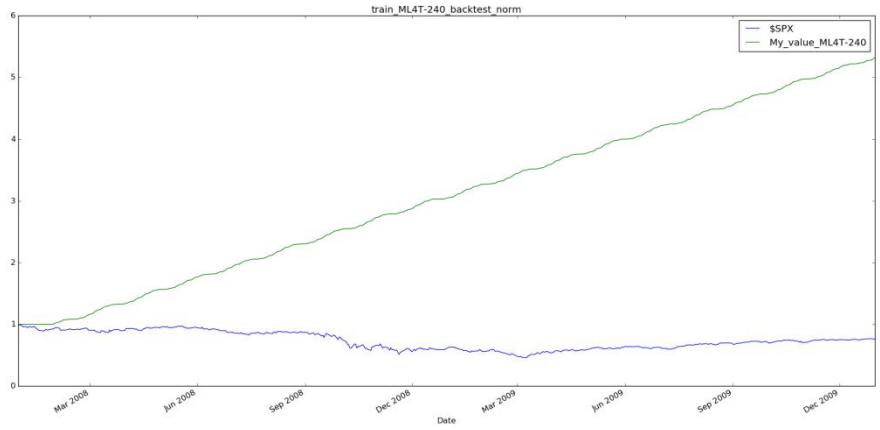

Chart 3 Sine data in-sample backtest

4.

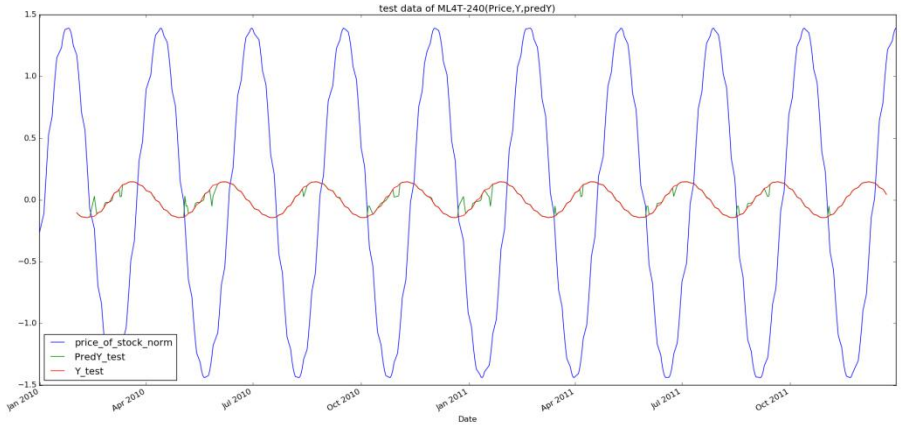

Chart 4 Sine data out-sample Training Y/Price/Predicted Y

5.

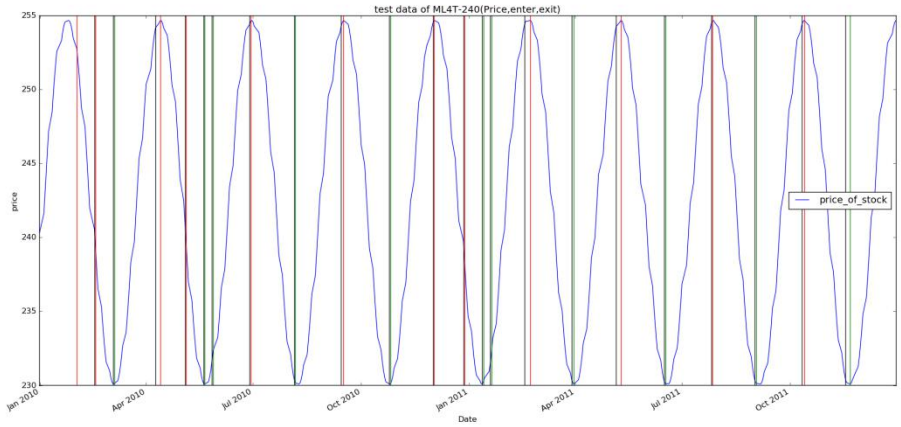

Chart 5 Sine data out-of-sample Entries/Exits

6.

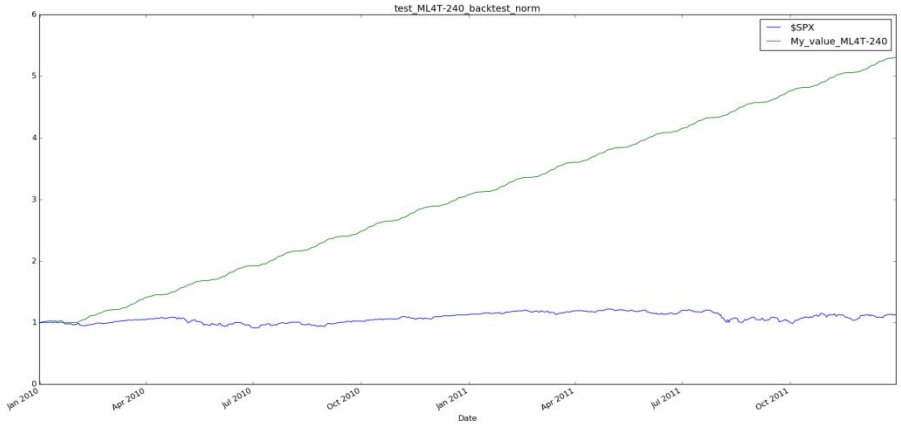

Chart 6 Sine data out-of-sample backtest

7.

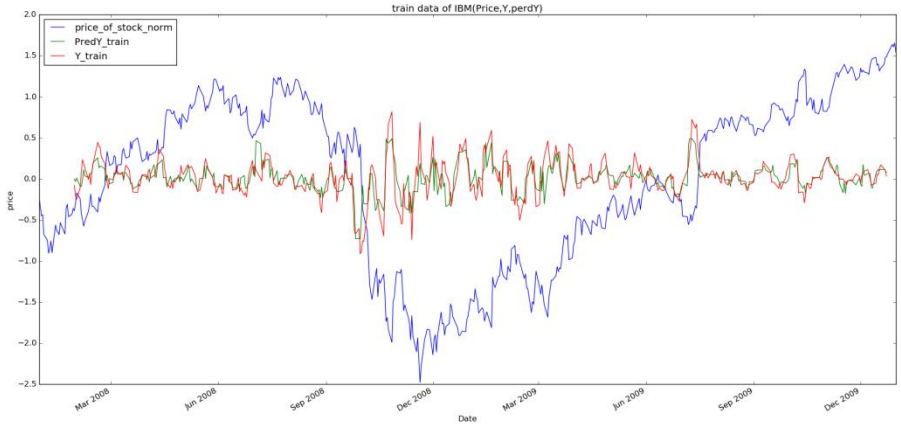

Chart 7 IBM data in-sample Training Y/Price/Predicted Y

8.

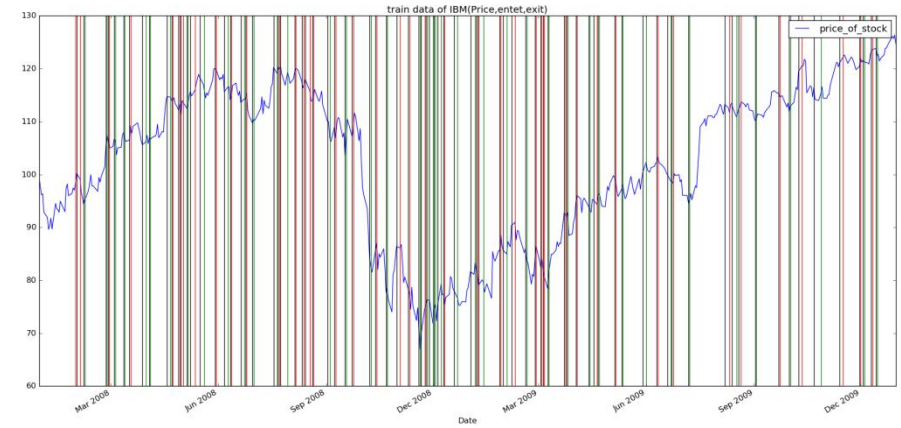

Chart 8 IBM data in-sample Entries/Exits

9.

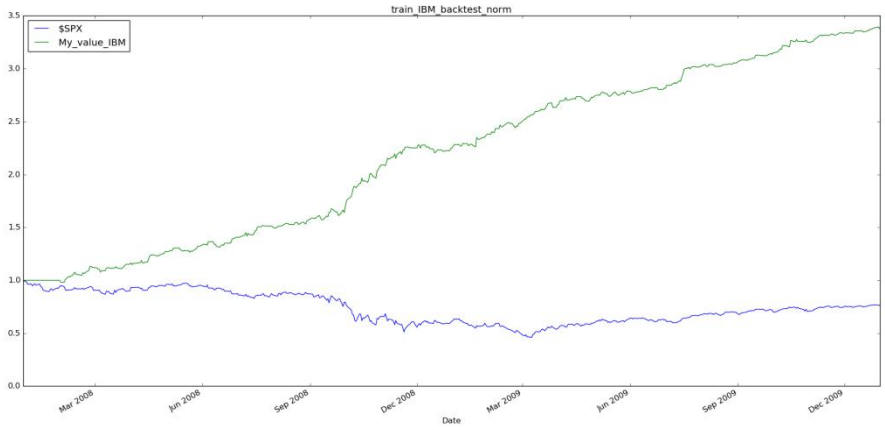

Chart 9 IBM data in-sample backtest

10.

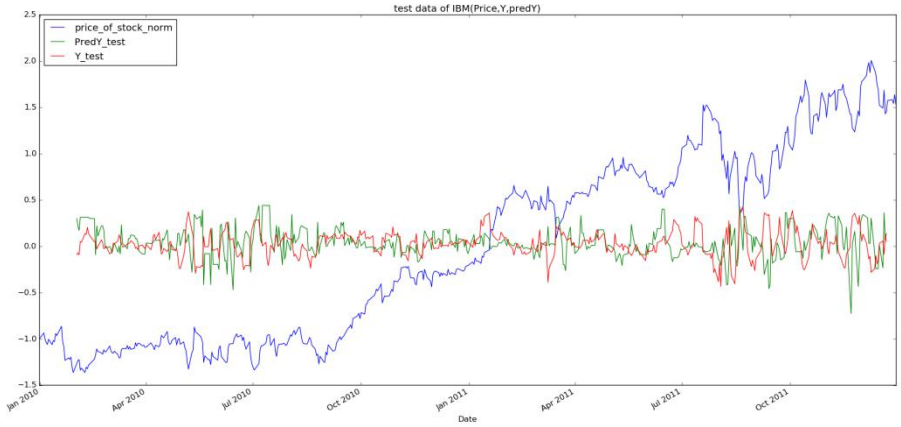

Chart 10 IBM data out-sample Training Y/Price/Predicted Y

11.

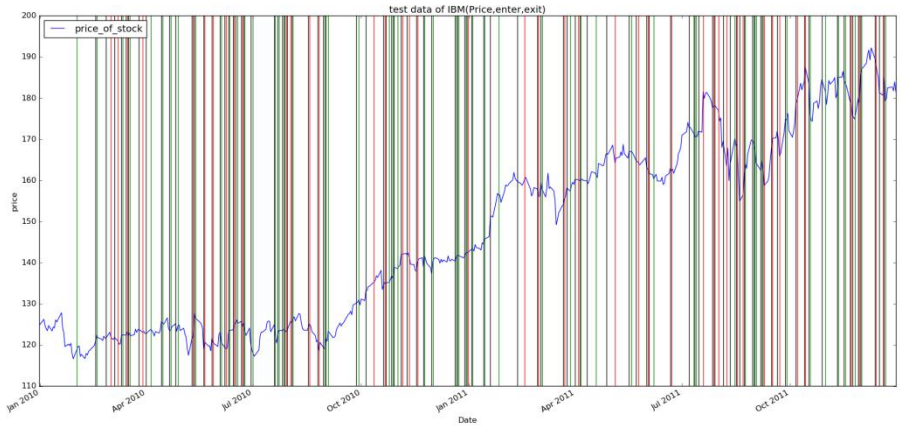

Chart 11 IBM data out-of-sample Entries/Exits

12.

Chart 12 IBM data out-of-sample backtest

Output:

Final values output:(we start with $10000)

Sine in-of-sample:53210.6911

Sine out-of-sample:53028.8802

IBM in-of-sample:33779.0

IBM out-of-sample:11964.0

Overview:

I use three features(normalized Bollinger Bands values, momentum and volatility) to predict the Y(5 day change in price). Then, according to the Y to make a a trading policy, it will output a .CSV file to tell you when to buy and sell. Finally, reading this file to simulate the market and get the final value, I can use it to compare with other stocks or policy or benchmark.

Describe each of the indicators you have selected in enough detail that someone else could reproduce them in code.

I use three features -- normalized Bollinger Bands values, momentum and volatility-- mentioned in the project4 introduction. Bollinger Bands are a volatility indicator similar to the Keltner channel. Momentum describe the direction during recent days. I use the following formulas to calculate them.

bb_value[t] = (price[t] - SMA[t])/(2 * stdev[t])

momentum[t] = (price[t]/price[t-N]) - 1

And Volatility is just the stdev of daily returns.

We use 20 days to calculate SMA and momentum. The stdev of daily returns use rolling std.

And I normalize these three feature to normal distribution. It aims to balance each feature affection. I use (feature-feature.mean)/feature.std to acquire normalized feature values.

Some of codes are as follow:

```
#get train data features
start_date = dt.datetime(2007,12,31)
end_date = dt.datetime(2009,12,31)
dfc = get_data(['ML4T-240'],pd.date_range(start_date,end_date))
priceC = dfc['ML4T-240']

#volatility
train_daily_rets = (priceC/priceC.shift(1))-1
train_volatility = pd.rolling_std(train_daily_rets,20)
#momentum
train_momentum = priceC/priceC.shift(19)-1.0
#bb_value
train_stdev = pd.rolling_std(priceC,20)
train_SMA = pd.rolling_mean(priceC,20)
train_bb_value = (train_stdev-train_SMA)/(2*train_stdev)

train_dataX = pd.concat([train_bb_value,train_momentum,train_volatility],keys=['train_BB','train_Momentum','train_Volatility'],axis = 1)

#normalize
train_dataX['train_BB'] = (train_dataX['train_BB']-train_dataX['train_BB'].mean())/train_dataX['train_BB'].std()
train_dataX['train_Momentum'] = (train_dataX['train_Momentum']-train_dataX['train_Momentum'].mean())/train_dataX['train_Momentum'].std()
train_dataX['train_Volatility'] = (train_dataX['train_Volatility']-train_dataX['train_Volatility'].mean())/train_dataX['train_Volatility'].std()
```

Describe your trading policy clearly.

I use similar way as project introduction, but they are different.

My policy is that when the predict Y is greater than 0.005, buy it, and then when predict Y is less than 0, I sell 100 shares; when the predict Y is less than -0.005, sell it, and then when predict Y is greater than 0, I buy back 100 shares.

Some of codes are as follow:

```
    if predY.ix[i][0] > 0.005 and shares == 0:
        writefile.writerow((predY.index[i],'ML4T-240','BUY','100'))
        shares = shares + 100
        i = i + 1
    elif predY.ix[i][0] > 0 and shares == -100:
        writefile.writerow((predY.index[i],'ML4T-240','BUY','0'))
        shares = shares + 100
        i = i + 1
    elif predY.ix[i][0] < 0 and shares == 100:
        writefile.writerow((predY.index[i],'ML4T-240','SELL','0'))
        shares = shares - 100
        i = i + 1
    elif predY.ix[i][0] < -0.005 and shares == 0:
        writefile.writerow((predY.index[i],'ML4T-240','SELL','-100'))
        shares = shares - 100
        i = i + 1
    else:i = i + 1
```

Discussion of results. Did it work well? Why? What would you do differently?

Comments on outputs(graph, final values):

According to graphs, in-sample sine and in-sample IBM backtests both perform very well, better than the manual policy I created for the last assignment and $SPX.

Out-of-sample sine backtest performs nearly identically as the in-sample test, it is also very well.

Out-of-sample IBM backtest perform not well, I guess it is because that my predict Y is not very

good, training data is too far to the testing data.
According to the final values of In-sample and Out-sample of IBM and Sine, we can earn money at least triple except for the last one after two years. In the last one, we can earn 20% of this in this investment, it is not bad.
The former three work well, the last one works not well.
Because the sine data is very easy to predict and learn, real data like IBM is more difficult to get its model without "rolling" model(that updates each day rolling forward). No matter how well policy is, it will perform not good without a accuracy predict Y. The data should update to get a real-time learner and predictor to predict Y as time goes. Maybe my trading policy can be improved.
The difficult parts are how to choose features and parameters of these features. Such as, I choose 20 days to calculate SMA and momentum, I need to compare the coherence and error between given data and prediction. Then I choose the parameters which perform best. Another difficult is to predict later two years data using former two years data, market is changing, we should update data that we already have known.