Stock Prediction with Hidden Markov Model

Letian Xu, Caroline (Fang) Cao

Abstract

Stock Markets are one of the most intricacy systems in the world, which are almost impossible to predict, and stock performances are an essential indicator of the strengths and weaknesses of the stock's corporation and the economy in general. Therefore, precisely predicting the stock price is of great research significance. This paper concerns four numeric attributes of the S&P 500 index as the research object, and the status and characteristics of the S&P 500 index are analyzed. We present two prediction approaches implementing Hidden Markov Model (HMM) to predict stock price. One is implemented with the Gaussian HMM and the other is with Multipolinimial HMM. For multinomial HMM implementation, we converted the stock price into binary classes "Price Decline" and "Price Increase" and this is a novel approach. Both approaches applied observation window for better performance and evaluations were conducted for comparing performance with different window sizes.

Keywords: Hidden Markov model; The stock price index; Prediction;

1. Introduction

Problem Statement

Stock investment is considered a high-risk financial activity. Generally, most investors fail to consider factors in stock price variation or do not master professional knowledge and experiences related to investment. To enhance the quality and profitability of the decision-making process for investors, different methods for predicting stock market prices have appeared. Therefore, how to select the appropriate and accurate prediction method, and how to effectively use the stock information to help investors make decisions has become a primary issue in stock investment.

Motivation

We have three main goals to achieve in this project. First, try to extract important information on stocks from their historical prices, including High, Low, Open, and Close price on every trading day's record. Then, using the Hidden Markov Model combined with transition matrix and emission matrix to predict future trends of stock or future performance on four stock index. Last, we want to testify if the Hidden Markov Model is able to predict stock price precisely and be sensitive to the financial crisis between 2006 and 2016 this particular period.

2. Related Work

For predicting the stock price of S&P 500, we referred to the methodologies presented by Nguyet[1] and Chen[2]. Both authors proved that the Hidden Markov Model is useful to predict stock price. However, the process of prediction of Nguyet[1] was limited on trying states less than 6. Also, Chen[2] didn't specify the number of states and both of them didn't present accuracy to evaluation. Hence, we improved the prediction process by adding determination steps to the best number of hidden states, and we further provided concise performance evaluation.

Chen[3] conducted data preprocessing steps to solve the missing label problem, and got high accuracy prediction performance on using Naive Bayes and Support Vector Machine methods. Although he conducted the process on a single company stock record, we refer to his preprocess method to use for our prediction of multinomial Hidden Markov Model.

Also, we referred to both Nguyet[1] and Gupta[5]'s work of initializing parameters for HMM. In our work. Gupta[5] conducted the data preprocessing steps to create two new numeric indexes, and we improved this preprocessing method for our prediction of multinomial Hidden Markov Model.

Zhang[4] inspired us with observation window methods and we implemented similarly in our process. Unlike Zhang [4], we didn't use the KNN to find the most similar observation sequence but find the most similar one based on the log likelihood.

3. Methodology

3.1 WorkFlow

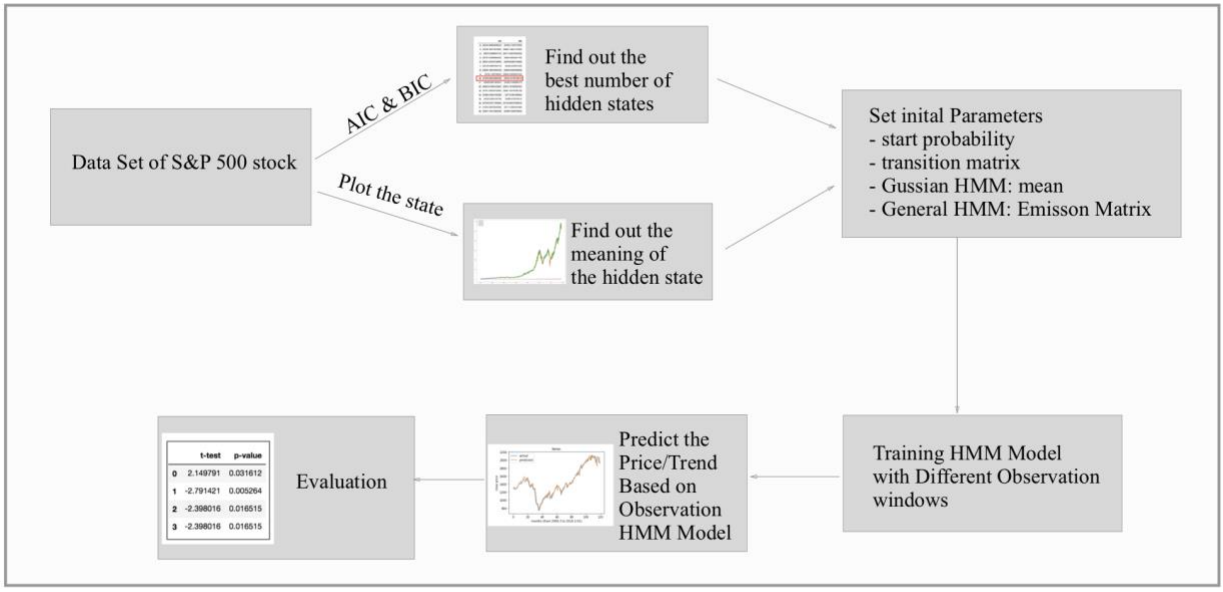

Figure 1. The whole workflow of predicting S&P 500 stock price with HMM

As shown in Figure 1 above, our whole workflow of predicting S&P 500 stock. We first downloaded and analyzed S&P 500 stock data. With the data, we implemented AIC and BIC to find out the best number of hidden states. To explore the meaning of hidden states, we plotted the hidden states into the price trend chart. With initializing the parameters, we trained HMM with different observation windows to predict the price or binary classes. In the end, we performed a concise evaluation.

3.2 Data

	Date	Open	High	Low	Close	Adj Close	Volume
0	1950-01-03	16.66	16.66	16.66	16.66	16.66	1260000
1	1950-01-04	16.85	16.85	16.85	16.85	16.85	1890000
2	1950-01-05	16.93	16.93	16.93	16.93	16.93	2550000

Table 1. The first three records of daily S&P 500 stock dataset

The original data contains S&P 500, the index stock of the US. benchmark market, trading daily and monthly records. There were 17320 daily trading records on S&P 500 from January 1950 to October 2018, 826 Monthly trading records on S&P 500 from January 1950 to October 2018. 7 original features included every trading day's date, the highest and lowest price in every trading day, the opening and closing price on every trading day, the adjusted closing price, and volume of every trading day.

3.2 Preprocessing

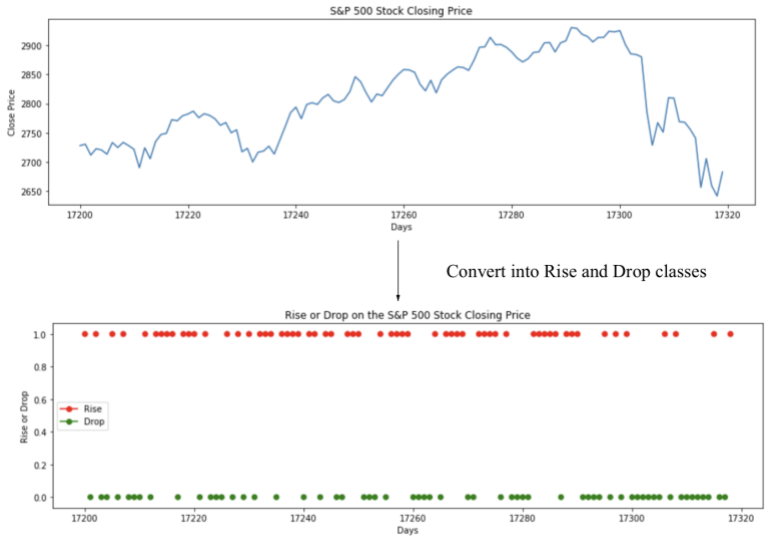

Figure 2. Recent 100 days' S&P 500 stock performance (6/11/2018 - 10/30/2018)

We checked raw data in the beginning, and find out the data was clean since there are no missing or duplicate values. For feature selection, Open, High, Low, Close is selected for learning from HMM based on the paper [1]. In order to predict the stock trend of price decline and price increase, we converted prices into these two trends by calculating the difference in every two days' close price. If the difference (Price of t+1 and Price of t) is negative, we converted this case into class "price decline"; vice versa. In this way, the prices of the data are converted into a two-dimensional nominal data for Multinomial HMM Approach.

3.3 Number of Hidden States Selection

In terms of selecting the best number of hidden states to build the HMM, we introduced the AIC and BIC to do the comparison between HMM models in different numbers of hidden states. Both information criteria are often used for comparing different models.

	AIC	BIC
2	[30452.496953848044]	[30482.17303779464]
3	[28766.786473679902]	[28826.138641573092]
4	[29072.30899673127]	[29171.229276553255]
5	[28787.255988684952]	[28935.63640841793]
6	[29051.205433739684]	[29258.93802136585]
7	[29149.046697952745]	[29426.0234814543]
8	[28292.196045930446]	[28648.30905328959]
9	[28194.1080768255]	[28639.249336024433]
10	[27909.286422962206]	[28453.34796198312]
11	[33909.63621603637]	[34562.51006286147]
12	[28840.618823440887]	[29612.197006052367]
13	[34761.229240702094]	[35661.403787082156]
14	[34680.436261555566]	[35719.0991996864]
15	[34202.33001245739]	[35389.3733703212]
16	[32783.676271009565]	[34128.992076588554]
17	[31604.058763340585]	[33117.53904461695]
18	[30907.194212982293]	[32598.73099793823]

Table 2. AIC and BIC score for selecting number of hidden states

The formula for calculating AIC and BIC score is below:

$AIC = -2\ln(L)+2k$

$BIC = -2\ln(L)+k*\ln(M)$

L: the likelihood calculated from the HMM model;
M: the number of observation points;
k: number of parameters N^2+2*N-1;
N: the number of states.

Then, we tested the HMM models in the range from 2 hidden states to 20 hidden states, and the best model in the group compared is the one that minimizes AIC and BIC scores, in both cases. From the table 2 above, we can find out the lowest AIC and BIC when the number of hidden states equal to 10. Hence, we took the 10 hidden states and 4 hidden states as the parameter to build the HMM model in the rest of the project.

3.4 Hidden State Interpretation

The Hidden Markov Model is a statistical model in which the system being modeled is assumed to be a Markov process with a particular number of hidden states. The transition between states is a Markov Process and can define by a matrix of state

transition probabilities. Also, the probability of observations given a state is determined by the emission probability.

Consider Ot be a vector of four elements-daily close, open, high and low, or a vector of two nominal performance, price rise and drop, and St to be the state on a different day express as t. The state on every day can be one of the assumed states. Figure 3 below shows a Hidden Markov Process with hidden state sequence and observation.

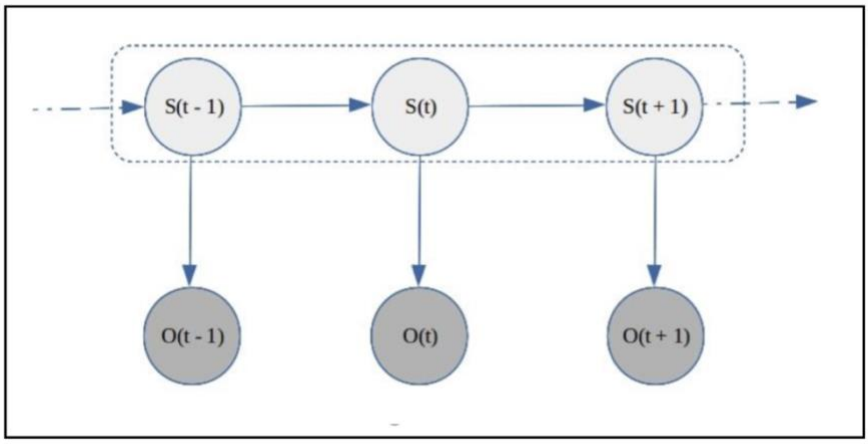

Figure 3. Hidden Markov Process with hidden state sequence and observation

Since the vector Ot represents real values, observations can regard as Multivariate Gaussian distributed. Also, the observations are assumed to independent, but the elements of an observation, like the close price in every day, may be correlated.

As time moving forward, the states keep changing as Markov Process with certain transition probability. Different combinations of hidden states based on its transition probability and emission probability will form its hidden states sequence.

The state sequence is not directly visible, and a variety of possible combinations. Hence, we apply the Viterbi Algorithm with the given model and observations; we can decode and get the hidden states sequence with the highest probability.

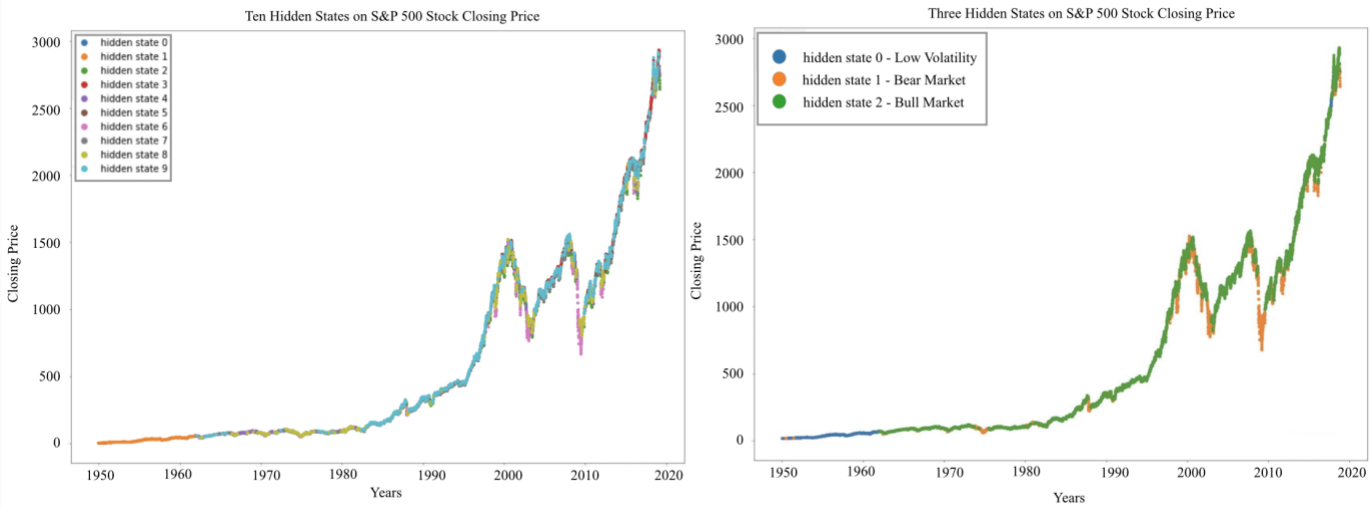

Figure 4: Hidden Sates Interpretation

In Figure 4 above, we plot the most likely hidden states sequence with the stock close price. In order to clarify the interpretation of the HMM, we conduct another HMM which select the number of hidden states equal to 3 to do the comparison.

In the HMM with 3 states, we can find out that the different state was generated from different stock's performance. Such the hidden state 1 with orange color, this state almost appeared every stock close price index downturn. Then we can assume that 3 hidden states represent "Bull Market", "Bear Market", and "Low Volatility Market".

Then, we can give some assumption on 10 hidden states based on corresponding observation performance. For example, hidden state 9 has the highest percentage of the period of rising stock index, we can assume that this state represents "No.1 Bull Market". And so on, We can assume the stock market performance of each hidden state according to the corresponding observed values.

3.5 Stock Price Prediction

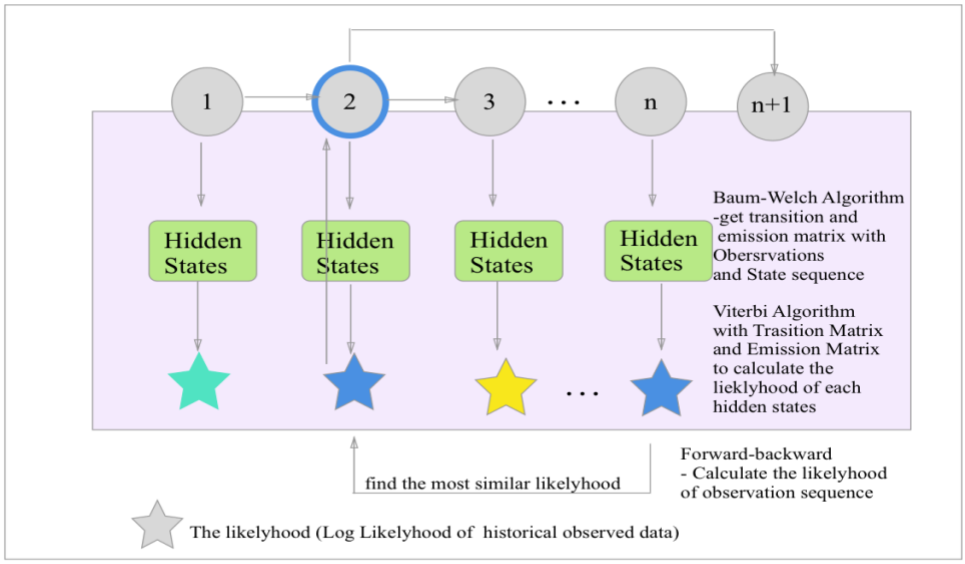

Figure 5. The overall process of finding out the most similar trading window for predicting the price of observation window n+1

We predicted the S&P 500 stock price based on this general process as shown in Figure 5 above. Based on each observation window (from 1 to n) and initialized model parameters, we can decode to find its best fit hidden state sequence with Viterbi Algorithm. Based on the observation sequence and state sequence, we calibrated HMM's parameters. With these calibrated parameters, we further calculate log likelihood of the observation sequence to find the most similar record from the history with Forward-backward Algorithm. With this record, we can predict the future price of period n+1. We clarify the whole process in detail in section 3.5.3.

3.5.1 Two HMM Approaches to Predict The Stock Price

First Approach

We used the numerical data (stock prices) for Gaussian HMM. It is different from general HMM, as the values in the emission matrix are generated from the Gaussian distribution of the numerical data based on each hidden state and observation.

Second Approach

We converted data into two classes: “price increase” and “price decline”, and used the Hidden Markov Model to predict binary classes with transition matrix and emission matrix. Then we were able to fix the unlabeled data problem and able to do classification.

We will present how we implement both approaches in detail in Section 3.5.2 and 3.5.3.

3.5.2 Parameters Initialization

First of all, we initialized the model parameters of the first date to predict the next day’s price. The basic elements of an HMM model are shown as the following:

$$\gamma \equiv \{A, B, p\}$$

where A is the transition Matrix, B is the emission matrix and the p is starting probability vector.

For initial transition matrix A, we calculated based on the number of states (N). For A=(a_{ij}), a_{ij}=1/N where a_{ij} is the value in matrix A (N by N matrix) under i^{th} row and j^{th}column; It means that we initialized with the same probability of transitioning from each hidden state. For the starting probability vector p, we set it to (1,0,...,0) assuming the 1st hidden state has the probability as 1 and the probabilities of all the other hidden states are 0; To initialize emission matrix B, gaussian HMM, and multinomial HMM adopt different approaches.

3.5.2.1 Gaussian HMM Parameter Initialization

For Gaussian HMM approach, the only difference from general HMM is that we assume the observation probability is continuously distributed and follows the Gaussian distribution. Thus, in the emission matrix B, the $b_{ij}$is the emission probability under hidden status i and observation j. The values in each row of matrix B stand for the emission probabilities of different observations from that hidden state. The values in

each column stand for emission probabilities of getting observation under all hidden states. These probabilities are based on the two parameters: mean (μ_i) and standard deviation (σ_i) of a historical observed data under each observation. For example, we calculated the mean and standard deviation of 10-year S&P 500 trading records for each observation: Open price, High price, Low price, and Close price. With passing these two parameters, we are able to calculate probabilities in the emission matrix based on Gaussian distribution.

3.5.2.2 Multinomial HMM Parameter Initialization

For Multinomial HMM approach, the basic method was followed by general HMM. The emission matrix B was generated from different observations' conditional probability. The value in matrix B represents the probability of a particular observation performance with a corresponding hidden state. In order to generate particular observation performance from the numeric stock index, we transfer the difference of every trading day's performance on close and volume into two discrete indexes: Rise. or Drop(in the data preprocessing step). With using these two indexes, we are able to calculate probabilities in the emission matrix with conditional probability.

3.5.3 Future Stock Price Prediction with Training Window

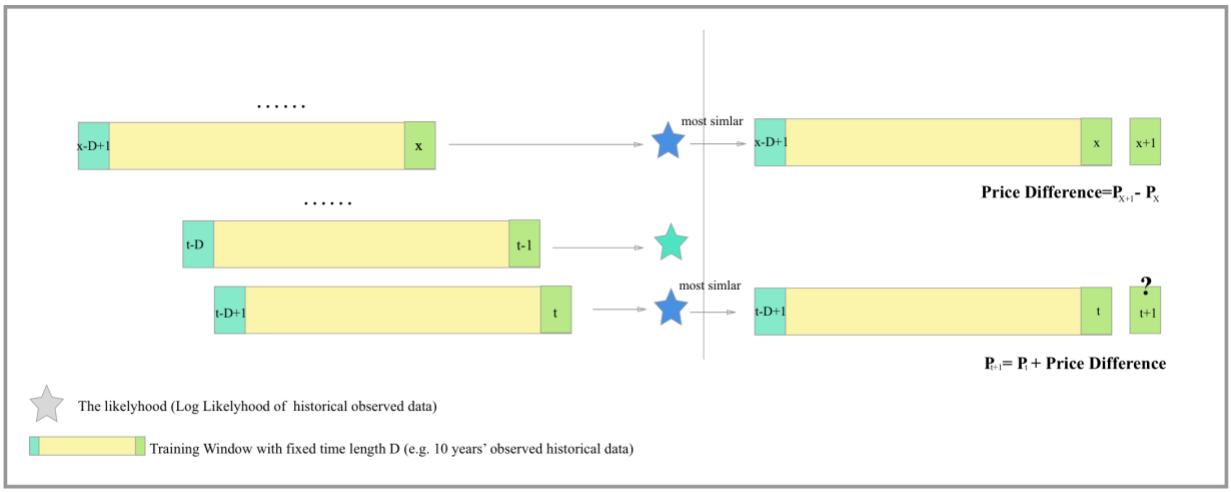

Figure 6. Predict Stock Price with HMM based on the training window

As shown in Figure 6 above, we used a training window with fixed time length D (for example, 10 years) to find the most similar observation sequence with the same time length. This approach highly increased our model's performance because the likelihood is not just based on the one-day observation but a certain time length. This is very different from the general classifier since the sequence of the observations is one of the most important features for the Hidden Markov model.

In detail, we are going to predict the stock price of t+1 (e.g. 2012-1-2), but we supposed we only know the historical data before t (e.g. 2012-1-1). Thus, we first trained the HMM model with the observed historical data (from t-D+1 to t) with initialized parameters and Forward Algorithm. As a result of training, we got a trained HMM model and log likelihood of the first observed historical data (from t-D+1 to t). With this same model, we go backward to use the older observed historical data (from t-D to t-1) for calculating its log likelihood with Forward Algorithm. We repeat the same process until we reach to the last date we set (e.g. 2002-1-1). We compared all the calculated log likelihoods and find out the most similar. If there are more than one most similar, we pick the latest one for prediction.

For the t+2 (e.g. 2012-1-3) prediction, we don't need to use the initialized parameters. Instead, from Baum-Welch algorithm with first observed historical dataset (from t-D+1 to

t), we had updated parameters for the HMM model of t+2. With these parameters passed, we repeat the same step for predict price of t+1 until the last date we want to predict its next day's price.

4. Results and Evaluation

In the Gaussian HMM approach, we tried from 10 months to 150 months and found the performance from 40 months to 120 months is significantly better as shown in Table 4 below. With these four different training windows (40 months, 60 months, 80 months and 120 months), we decided that the performance of 80 months' training window is the best based on the three criteria (accuracy, sensitivity, and specificity).

Training Window	Accuracy	Sensitivity	Specificity
40 months	0.51	0.57	0.50
60 months	0.54	0.59	0.52
80 months	0.65	0.62	0.63
120 months	0.64	0.61	0.59

Table 4. Gaussian HMM approach with different window sizes

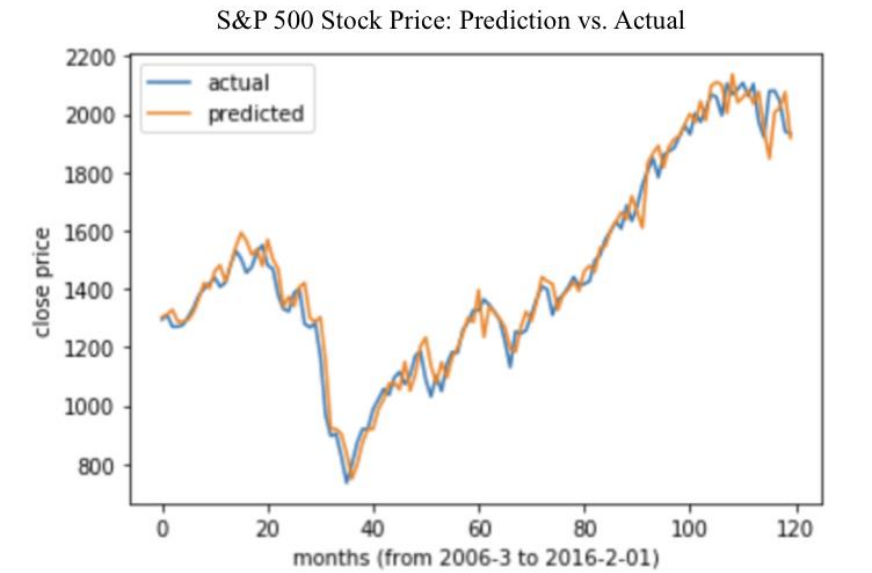

Figure 7. Comparing actual price with price predicted from Gaussian HMM

As shown in Figure 7 above, we visualized one of our predictions by comparing the actual price. Regarding our motivation, we selected the period (from 2006-March to 2016-February) including the recent well known financial crisis to see if we can predict the financial crisis or not. From the figure, We can tell our model is sensitive the financial crisis. We also calculated the RMSE (root-mean-square error) is around 60, based on the difference between actual and predicted price. From this criteria, we can see the predicted error is relatively acceptable for stock market since this is a monthly price prediction. Furthermore, the actual price is from 800 US dollars to 2200 US dollars and 60 US dollars error is relatively less than 7.5% of the lowest price.

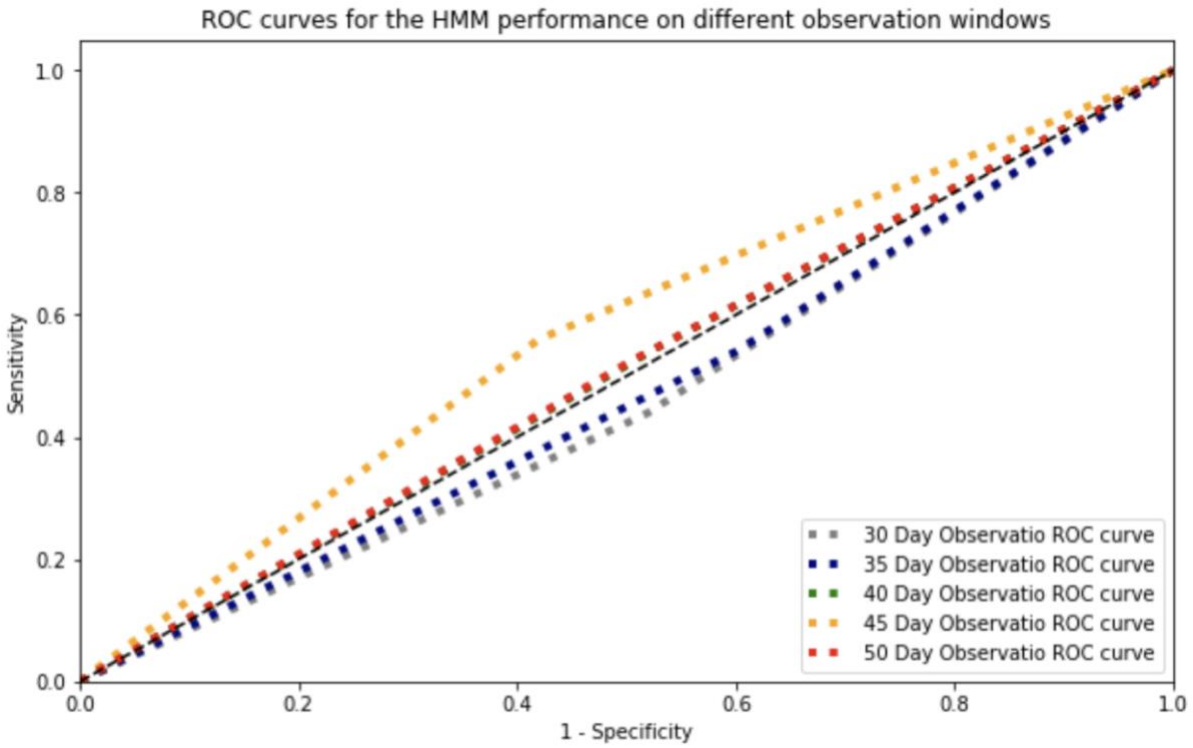

Figure 8. ROC curve for Multinomial HMM perfomace on different observation windows

In the second approach, the ROC curve in Figure 8 above was introduced to find the best prediction model. Then, the prediction model with 45 days' observation window was selected among other models. 56% overall accuracy, and 62% specificity with the positive label as "Rise". In general, the second approach did a better job on predicting the future if stock price will not rise.

	Accuracy	Sensitivity	Specificity
5 Day OB Prediction	**0.48**	**0.44**	**0.51**
10 Day OB Prediction	**0.51**	**0.43**	**0.58**
15 Day OB Prediction	**0.5**	**0.45**	**0.54**
20 Day OB Prediction	**0.51**	**0.44**	**0.57**
25 Day OB Prediction	**0.48**	**0.46**	**0.50**
30 Day OB Prediction	**0.5**	**0.45**	**0.54**
35 Day OB Prediction	**0.555**	**0.48**	**0.61**
40 Day OB Prediction	**0.44**	**0.37**	**0.50**
45 Day OB Prediction	**0.56**	**0.48**	**0.62**
50 Day OB Prediction	**0.46**	**0.46**	**0.45**

Table 5. Performance of Prediction with Multinomial HMM and different observation window sizes

In order to testify if the best model was generated by random instance, we build up the null hypothesis and conduct the t-test to check if there was no difference between different model's results. All the p-value was less than 0.05, then we can say the best result is significantly different from other results.

5. Limitation

The first limitation is that we can predict tomorrow as a single day, but cannot predict next whole month's price trend with our model. We are not able to predict the future which is too far from now since we never know the future price and we are not able to generate the observation sequence.

Also, in the ROC curve of the evaluation part, we find out there was a boundary in the second approach. As we plot the different prediction model with the corresponding observation window, several model's results were worse than random classification. Hence, further analysis was carried out to find the critical point for the number of days to use the observation window in the prediction.

We find out that the prediction model with observation window less than 40 days will cause the bad result, which is worse than random. Then, we assume that training observation window needs at least 40 days in the second approach. Otherwise, the smaller the number of observation days used for training window, the more unstable the model.

6. Conclusion

Stock performances are an essential indicator of the strengths and weaknesses of the stock's corporation and the economy in general.
With the first approach while using HMM, our prediction was precise to capture the economic crisis in the past 12 years. With the second approach while using HMM, we were able to find out the most likely tomorrow stock performance.

We figured that we are able to interpret the meaning of hidden status by plotting them on the stock price trend. It will be helpful when using this HMM model for further deep learning.

7. Future Work

We need to improve HMM and enable it to capture the volatility of the stock prices, instead of only divide the stock into ‘rise’ and ‘drop’ two situations. And the predictions were limited when prices are predicted for more than one day. We can extend the prediction to a form of density function rather than original continuous index values. This way can we output richer information about the market movements.

Also, In the observed 10 hidden states, we need to estimate them to get the best characteristic for different assets at different states. In the future, we still need to apply HMM on other datasets to testify HMM if it is useful and efficient on other machine learning problems.

Peer Evaluation:

Score: 100/100
My partner is Caroline Cao. She has a thorough understanding of data science. She is very active in the whole project and has undertaken a lot of work in the Gaussian HMM. Approach.

Letian Xu

Reference:

[1]Nguyet Nguyen, 2018. "Hidden Markov Model for Stock Trading," International Journal of Financial Studies, MDPI, Open Access Journal, vol. 6(2), pages 1-17, March.

[2]HE Fengxia, HUANG Jingfeng. Prediction of Stock Price Index with Hidden Markov Model[EB/OL]. Beijing: Science paper Online (2016-04-15). http://www.paper.edu.cn/releasepaper/content/201604-180.

[3]Chen, Y.-J., Chen, Y.-M., Lu, C.L.: Enhancement of stock market forecasting using an improved fundamental analysis-based approach. Springer, Heidelberg (2016)

[4]Zhang, Xi; Li, Yixuan; Wang, Senzhang; Fang, Binxing; Yu, Philip S. Enhancing Stock Market Prediction with Extended Coupled Hidden Markov Model over Multi-Sourced Data, eprint arXiv:1809.00306, 09/2018

[5]A. Gupta “Stock market prediction using Hidden Markov Models”, IEEE Engineering and Systems(SCES), 2012 Students Conference on, pp.1-4,2012.

[6]Nguyen, Nguyet, and Dung Nguyen. 2015. Hidden Markov Model for Stock Selection. Risks 3: 455–73.

Appendix

```
# Process of AIC & BIC
pd.options.mode.use_inf_as_na = True

NUM_TEST = 100
K = 50
NUM_ITERS=10000
DEFAULT_TOL = 1e-12


likelihood_vect = np.empty([0,1])
aic_vect = np.empty([0,1])
bic_vect = np.empty([0,1])

STATE_SPACE = range(2,19,1)
csv_data = pd.read_csv('BABA.csv')


del(csv_data['Date'])
del(csv_data['Adj Close'])

csv_data.head()

csv_data['Volume'] = (csv_data['Volume'] / 1000).astype(int) / 100
csv_data.head()

def get_exp_preprocessing(df,alpha=0.9):
    edata = df.ewm(alpha=alpha).mean()
    return edata

dataset = get_exp_preprocessing(csv_data)
dataset.head()

predicted_stock_data = np.empty([0,dataset.shape[1]])
likelihood_vect = np.empty([0,1])
aic_vect = np.empty([0,1])
bic_vect = np.empty([0,1])

dataset = np.array(dataset)

for states in STATE_SPACE:
    print(states)
    num_params = states**2 + states
    dirichlet_params_states = np.random.randint(1,50,states)
    #model = hmm.GaussianHMM(n_components=states, covariance_type='full', startprob_prior=dirichlet_params_states,
transmat_prior=dirichlet_params_states, tol=0.0001, n_iter=NUM_ITERS, init_params='mc')
    model = hmm.GaussianHMM(n_components=states, covariance_type='full', tol=DEFAULT_TOL, n_iter=NUM_ITERS)
    model.fit(dataset[NUM_TEST:,:])
    if model.monitor_.iter == NUM_ITERS:
        print('Increase number of iterations')
        sys.exit(1)
    likelihood_vect = np.vstack((likelihood_vect, model.score(dataset)))
    aic_vect = np.vstack((aic_vect, -2 * model.score(dataset) + 2 * num_params))
    bic_vect = np.vstack((bic_vect, -2 * model.score(dataset) +  num_params * np.log(dataset.shape[0])))

opt_states = np.argmin(bic_vect) + 2
print('Optimum number of states are {}'.format(opt_states))

df = pd.DataFrame(list(zip(aic_vect,bic_vect)),
                  columns=("AIC","BIC"),
                 index = np.arange(2,20,1))
df

opt_states = 10

# Data Preprocessing: Transfer to Binary Situation
```

```
df = pd.read_csv('SP500.csv')

# Transfer column to list
date_list = pd.to_datetime(df['Date'])
volume_list = df['Volume']
open_list = df['Volume']
close_list = df['Close']

# Transfer everyday's closing prices' difference in percentage to Binary Variable
feature_1=np.diff(close_list)/close_list[:-1]
rise_or_drop = []

for i in range(0,17319):
    if feature_1[i] > 0:
        rise_or_drop.append(1)
    else: rise_or_drop.append(0)

# Transfer everyday's volume prices' difference in percentage to Binary Variable
feature_2=np.diff(volume_list)/volume_list[:-1]
more_or_less = []

for i in range(0,17319):
    if feature_2[i] > 0:
        more_or_less.append(1)
    else: more_or_less.append(0)

# Zip in a two-dimensional dataframe
x = pd.DataFrame(list(zip(rise_or_drop,more_or_less)),
             columns = ['Rise or Drop','More or Less'],
            index = np.arange(1,17320,1))

# Binary Variable Plot in recent 120 days
plt.figure(figsize=(15,4))
plt.title("Rise or Drop on the S&P 500 Stock Closing Price")
color= ['red' if i == 1 else 'green' for i in rise_or_drop[17200:17321]]
plt.scatter(np.arange(17200,17319,1),rise_or_drop[17200:17321],c=color)
plt.xlabel("Days")
plt.ylabel("Rise or Drop")
legend_elements = [Line2D([0], [0], color='r',marker='o',label='Rise'),
                   Line2D([0], [0], color='g',marker='o', label='Drop')]
plt.legend(handles=legend_elements,loc="center left")
plt.show()

# Closing Price Plot in recent 120 days
plt.figure(figsize=(15,4))
plt.plot(close_list[17200:17321])
plt.title("S&P 500 Stock Closing Price")
plt.xlabel("Days")
plt.ylabel("Close Price")
plt.show()

# Set up with training and testing datasets
# First 56 years data which alomost 80% data will be training and rest 10 years will be testing
training = x[0:13670]
testing = x[13669:17320]
testing = testing.set_index(np.arange(1,3651,1))

# build HMM model
from hmmlearn.hmm import MultinomialHMM
model=hmm.MultinomialHMM(n_components=10, tol=1e-12, n_iter=1000)


# ## Traning Result

# Training Result
model = model.fit(training)
observe_window = np.arange(5,55,5)

for K in observe_window:
```

```
    iters = 0;
    past_likelihood = []
    while iters < 13670 - K:
        past_likelihood = np.append(past_likelihood, model.score(np.flipud(training[iters:iters + K])))
        iters = iters + 1

    likelihood_diff_idx_1 = []
    past_similar = []

    for i in range(13670 - K - 1,0,-1):
        curr_likelihood = model.score(training[i:i+K])
        likelihood_diff_idx = np.argmin(np.absolute(past_likelihood - curr_likelihood))
        likelihood_diff_idx_1.append(likelihood_diff_idx+K+1)
        past_similar.append(rise_or_drop[likelihood_diff_idx+K+1])

    cm = confusion_matrix(rise_or_drop[K:100+K],list(reversed(past_similar))[0:100])
    accuracy1=(cm[0,0]+cm[1,1])/sum(sum(cm))
    sensitivity1 = cm[0,0]/(cm[0,0]+cm[0,1])
    specificity1 = cm[1,1]/(cm[1,0]+cm[1,1])

    print("The Accuracy of Training set by 10-HMM prediction on", K, "days obeservation is", accuracy1)
    print("The Sensitivity of Training set by 10-HMM prediction on", K, "days obeservation is", sensitivity1)
    print("The Specificity of Training set by 10-HMM prediction on", K, "days obeservation is", specificity1)


# ## Testing

model = model.fit(testing)
observe_window = np.arange(5,55,5)

for K in observe_window:

    iters = 0;
    past_likelihood = []
    while iters < 3650 - K:
        past_likelihood = np.append(past_likelihood, model.score(np.flipud(testing[iters:iters + K])))
        iters = iters + 1

    likelihood_diff_idx_1 = []
    past_similar_test = []

    for i in range(3650 - K - 1,0,-1):
        curr_likelihood = model.score(testing[i:i+K])
        likelihood_diff_idx = np.argmin(np.absolute(past_likelihood - curr_likelihood))
        likelihood_diff_idx_1.append(likelihood_diff_idx+K+1)
        past_similar_test.append(rise_or_drop[likelihood_diff_idx+K+1])

    cm = confusion_matrix(list(reversed(rise_or_drop))[0:200],past_similar_test[0:200])
    accuracy1=(cm[0,0]+cm[1,1])/sum(sum(cm))
    sensitivity1 = cm[0,0]/(cm[0,0]+cm[0,1])
    specificity1 = cm[1,1]/(cm[1,0]+cm[1,1])

    print("The Accuracy of Testing set by 10-HMM prediction on", K, "days obeservation is", accuracy1)
    print("The Sensitivity of Testing set by 10-HMM prediction on", K, "days obeservation is", sensitivity1)
    print("The Specificity of Testing set by 10-HMM prediction on", K, "days obeservation is", specificity1)

from sklearn import metrics
from sklearn.metrics import accuracy_score

K = 50
iters = 0;
past_likelihood = []
while iters < 3650 - K:
    past_likelihood = np.append(past_likelihood, model.score(np.flipud(testing[iters:iters + K])))
    iters = iters + 1

likelihood_diff_idx_1 = []
past_similar_test = []

for i in range(3650 - K - 1,0,-1):
```

```
    curr_likelihood = model.score(testing[i:i+K])
    likelihood_diff_idx = np.argmin(np.absolute(past_likelihood - curr_likelihood))
    likelihood_diff_idx_1.append(likelihood_diff_idx+K+1)
    past_similar_test.append(rise_or_drop[likelihood_diff_idx+K+1])

cm = confusion_matrix(list(reversed(rise_or_drop))[0:200],past_similar_test[0:200])
accuracy1=(cm[0,0]+cm[1,1])/sum(sum(cm))
sensitivity1 = cm[0,0]/(cm[0,0]+cm[0,1])
specificity1 = cm[1,1]/(cm[1,0]+cm[1,1])

y_test = list(reversed(rise_or_drop))[0:100]
y_test_score = past_similar_test[0:100]

fpr_50, tpr_50, thr_50 = metrics.roc_curve(y_test,y_test_score)

K = 45
iters = 0;
past_likelihood = []
while iters < 3650 - K:
    past_likelihood = np.append(past_likelihood, model.score(np.flipud(testing[iters:iters + K])))
    iters = iters + 1

likelihood_diff_idx_1 = []
past_similar_test = []

for i in range(3650 - K - 1,0,-1):
    curr_likelihood = model.score(testing[i:i+K])
    likelihood_diff_idx = np.argmin(np.absolute(past_likelihood - curr_likelihood))
    likelihood_diff_idx_1.append(likelihood_diff_idx+K+1)
    past_similar_test.append(rise_or_drop[likelihood_diff_idx+K+1])

cm = confusion_matrix(list(reversed(rise_or_drop))[0:200],past_similar_test[0:200])
accuracy1=(cm[0,0]+cm[1,1])/sum(sum(cm))
sensitivity1 = cm[0,0]/(cm[0,0]+cm[0,1])
specificity1 = cm[1,1]/(cm[1,0]+cm[1,1])

y_test = list(reversed(rise_or_drop))[0:100]
y_test_score = past_similar_test[0:100]

fpr_45, tpr_45, thr_45 = metrics.roc_curve(y_test,y_test_score)

K = 40
iters = 0;
past_likelihood = []
while iters < 3650 - K:
    past_likelihood = np.append(past_likelihood, model.score(np.flipud(testing[iters:iters + K])))
    iters = iters + 1

likelihood_diff_idx_1 = []
past_similar_test = []

for i in range(3650 - K - 1,0,-1):
    curr_likelihood = model.score(testing[i:i+K])
    likelihood_diff_idx = np.argmin(np.absolute(past_likelihood - curr_likelihood))
    likelihood_diff_idx_1.append(likelihood_diff_idx+K+1)
    past_similar_test.append(rise_or_drop[likelihood_diff_idx+K+1])

cm = confusion_matrix(list(reversed(rise_or_drop))[0:200],past_similar_test[0:200])
accuracy1=(cm[0,0]+cm[1,1])/sum(sum(cm))
sensitivity1 = cm[0,0]/(cm[0,0]+cm[0,1])
specificity1 = cm[1,1]/(cm[1,0]+cm[1,1])

y_test = list(reversed(rise_or_drop))[0:100]
y_test_score = past_similar_test[0:100]

fpr_40, tpr_40, thr_40 = metrics.roc_curve(y_test,y_test_score)

K = 35
iters = 0;
past_likelihood = []
```

```
while iters < 3650 - K:
    past_likelihood = np.append(past_likelihood, model.score(np.flipud(testing[iters:iters + K])))
    iters = iters + 1

likelihood_diff_idx_1 = []
past_similar_test = []

for i in range(3650 - K - 1,0,-1):
    curr_likelihood = model.score(testing[i:i+K])
    likelihood_diff_idx = np.argmin(np.absolute(past_likelihood - curr_likelihood))
    likelihood_diff_idx_1.append(likelihood_diff_idx+K+1)
    past_similar_test.append(rise_or_drop[likelihood_diff_idx+K+1])

cm = confusion_matrix(list(reversed(rise_or_drop))[0:200],past_similar_test[0:200])
accuracy1=(cm[0,0]+cm[1,1])/sum(sum(cm))
sensitivity1 = cm[0,0]/(cm[0,0]+cm[0,1])
specificity1 = cm[1,1]/(cm[1,0]+cm[1,1])

y_test = list(reversed(rise_or_drop))[0:100]
y_test_score = past_similar_test[0:100]

fpr_35, tpr_35, thr_35 = metrics.roc_curve(y_test,y_test_score)

K = 30
iters = 0;
past_likelihood = []
while iters < 3650 - K:
    past_likelihood = np.append(past_likelihood, model.score(np.flipud(testing[iters:iters + K])))
    iters = iters + 1

likelihood_diff_idx_1 = []
past_similar_test = []

for i in range(3650 - K - 1,0,-1):
    curr_likelihood = model.score(testing[i:i+K])
    likelihood_diff_idx = np.argmin(np.absolute(past_likelihood - curr_likelihood))
    likelihood_diff_idx_1.append(likelihood_diff_idx+K+1)
    past_similar_test.append(rise_or_drop[likelihood_diff_idx+K+1])

cm = confusion_matrix(list(reversed(rise_or_drop))[0:200],past_similar_test[0:200])
accuracy1=(cm[0,0]+cm[1,1])/sum(sum(cm))
sensitivity1 = cm[0,0]/(cm[0,0]+cm[0,1])
specificity1 = cm[1,1]/(cm[1,0]+cm[1,1])

y_test = list(reversed(rise_or_drop))[0:1000]
y_test_score = past_similar_test[0:1000]

fpr_30, tpr_30, thr_30 = metrics.roc_curve(y_test,y_test_score)


K = 25
iters = 0;
past_likelihood = []
while iters < 3650 - K:
    past_likelihood = np.append(past_likelihood, model.score(np.flipud(testing[iters:iters + K])))
    iters = iters + 1

likelihood_diff_idx_1 = []
past_similar_test = []

for i in range(3650 - K - 1,0,-1):
    curr_likelihood = model.score(testing[i:i+K])
    likelihood_diff_idx = np.argmin(np.absolute(past_likelihood - curr_likelihood))
    likelihood_diff_idx_1.append(likelihood_diff_idx+K+1)
    past_similar_test.append(rise_or_drop[likelihood_diff_idx+K+1])

cm = confusion_matrix(list(reversed(rise_or_drop))[0:200],past_similar_test[0:200])
accuracy1=(cm[0,0]+cm[1,1])/sum(sum(cm))
sensitivity1 = cm[0,0]/(cm[0,0]+cm[0,1])
specificity1 = cm[1,1]/(cm[1,0]+cm[1,1])
```

```
y_test = list(reversed(rise_or_drop))[0:200]
y_test_score = past_similar_test[0:200]

fpr_25, tpr_25, thr_25 = metrics.roc_curve(y_test,y_test_score)

# Roc Curve
plt.figure(figsize=(10,6))
plt.plot(fpr_40, tpr_40,
        label='30 Day Observatio ROC curve',
        color='grey', linestyle=':', linewidth=4)

plt.plot(fpr_50, tpr_50,
        label='35 Day Observatio ROC curve',
        color='navy', linestyle=':', linewidth=4)

plt.plot(fpr_30, tpr_30,
        label='40 Day Observatio ROC curve',
        color='green', linestyle=':', linewidth=4)

plt.plot(fpr_35, tpr_35,
        label='45 Day Observatio ROC curve',
        color='orange', linestyle=':', linewidth=4)

plt.plot(fpr_45, tpr_45,
        label='50 Day Observatio ROC curve',
        color='red', linestyle=':', linewidth=4)


plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('1 - Specificity')
plt.ylabel('Sensitivity')
plt.title('ROC curves for the HMM performance on different observation windows')
plt.legend(loc="lower right")
plt.show()

K = 1
iters = 0;
past_likelihood = []
while iters < 3650 - K:
    past_likelihood = np.append(past_likelihood, model.score(np.flipud(testing[iters:iters + K])))
    iters = iters + 1

likelihood_diff_idx_1 = []
past_similar_test = []

for i in range(3650 - K - 1,0,-1):
    curr_likelihood = model.score(testing[i:i+K])
    likelihood_diff_idx = np.argmin(np.absolute(past_likelihood - curr_likelihood))
    likelihood_diff_idx_1.append(likelihood_diff_idx+K+1)
    past_similar_test.append(rise_or_drop[likelihood_diff_idx+K+1])

cm = confusion_matrix(list(reversed(rise_or_drop))[0:200],past_similar_test[0:200])
accuracy1=(cm[0,0]+cm[1,1])/sum(sum(cm))
sensitivity1 = cm[0,0]/(cm[0,0]+cm[0,1])
specificity1 = cm[1,1]/(cm[1,0]+cm[1,1])

y_test = list(reversed(rise_or_drop))[0:200]
y_test_score_ = past_similar_test[0:3000]

from scipy import stats
y_test = list(reversed(rise_or_drop))[0:3000]

df = pd.DataFrame(list(zip(y_test_score,y_test_score_30)),
                  columns=['45 OB','30 OB'])


# In[135]:
```

```
P_Value = []
T_Test = []

t_test, p_value = stats.mstats.ttest_ind(y_test_score,y_test_score_)
P_Value.append(p_value)
T_Test.append(t_test)
print("Testing Set: t_statistic is %0.3f and p-value is %0.6f"% (t_stat, p_value))

df = pd.DataFrame(list(zip(T_Test,P_Value)),
                  columns=['t-test','p-value'])
df

#Calculate the t-test statistics and p-value for testing set
t_stat, p_value = stats.ttest_ind(y_test_score,y_test_score_30)
print("Testing Set: t_statistic is %0.3f and p-value is %0.6f"% (t_stat, p_value))

df = pd.DataFrame(list(zip(y_test_score,y_test_score_35)),
                  columns=['45 OB','35 OB'])

#Calculate the t-train statistics and p-value for training set
t_stat, p_value = stats.ttest_ind(y_test_score,y_test_score_35)
print("training Set: t_statistic is %0.3f and p-value is %0.6f"% (t_stat, p_value))

df = pd.DataFrame(list(zip(y_test_score,y_test_score_40)),
                  columns=['45 OB','40 OB'])

#Calculate the t-test statistics and p-value for testing set
t_stat, p_value = stats.ttest_ind(y_test_score,y_test_score_40)
print("Testing Set: t_statistic is %0.3f and p-value is %0.6f"% (t_stat, p_value))

df = pd.DataFrame(list(zip(y_test_score,y_test_score_50)),
                  columns=['45 OB','50 OB'])

#Calculate the t-test statistics and p-value for testing set
t_stat, p_value = stats.ttest_ind(y_test_score,y_test_score_50)
print("Testing Set: t_statistic is %0.3f and p-value is %0.6f"% (t_stat, p_value))

#Process for Ganssian HMM

# coding: utf-8

# In[1]:


get_ipython().magic(u'matplotlib inline')

import warnings
import time
import sys
import numpy as np
import matplotlib.pyplot as plt
from hmmlearn import hmm

import random
import pandas as pd

pd.options.mode.use_inf_as_na = True
warnings.filterwarnings("ignore", category=DeprecationWarning)


# In[2]:


stock = "sp500_monthly"
# dataset = np.genfromtxt('../Dataset/' + stock + '.csv', delimiter=',')
csv_data = pd.read_csv('../Dataset/' + stock + '.csv')

del(csv_data['Adj Close'])
```

```
del(csv_data['Volume'])

csv_data.head()


# In[3]:


df = pd.DataFrame(csv_data)
df_month=df.groupby(pd.DatetimeIndex(df['Date']).to_period('M')).nth(0)


# In[4]:


def predictPrice(df_month,start_date, predicted_date,current_date,
transmat_retune_prior=None,means_retune_prior=None,startprob_retune_prior=None,covars_retune_prior=None):
    mask = (df_month['Date'] >=start_date) & (df_month['Date'] < predicted_date)
    df_1stBlock_train=df_month.loc[mask]
    # print df_1stBlock_train
    df_1stBlock_train.reset_index(drop=True, inplace=True)
    del df_1stBlock_train['Date']

    model = hmm.GaussianHMM(n_components=4, init_params="")
#     if covars_retune_prior is None:
#         model = hmm.GaussianHMM(n_components=4, covariance_type='full', init_params="")

    state_number=4

    if transmat_retune_prior is None:
        model.transmat_ = np.full((state_number, state_number), 1.0/state_number)
    else:
        model.transmat_ = transmat_retune_prior

    if means_retune_prior is None:
        u=[]
        OpenMean=df_1stBlock_train['Open'].mean()
        HighMean=df_1stBlock_train['High'].mean()
        LowMean=df_1stBlock_train['Low'].mean()
        CloseMean=df_1stBlock_train['Close'].mean()
        for i in range(state_number):
            state_vector=[]
            state_vector.append(OpenMean+np.random.normal(0, 1))
            state_vector.append(HighMean+np.random.normal(0, 1))
            state_vector.append(LowMean+np.random.normal(0, 1))
            state_vector.append(CloseMean+np.random.normal(0, 1))
            #print state_vector
            u.append(state_vector)
        model.means_ = u
    else:
        model.means_ = means_retune_prior

    if startprob_retune_prior is None:
        sratprob=np.zeros(state_number)
        sratprob[0]=1.0
        model.startprob_ = sratprob
    else:
        model.startprob_ = startprob_retune_prior

#     if covars_retune_prior is not None:
#         model.covars_ = covars_retune_prior

    model.fit(df_1stBlock_train)
    transmat_retune_prior=model.transmat_
    means_retune_prior =model.means_
    startprob_retune_prior=model.startprob_
    covars_retune_prior=model.covars_

    current_score=model.score(df_1stBlock_train)
```

```
    # backward to find the closest item
    backward_list = df_month.loc[(df_month['Date'] < predicted_date)][:-2]
    actualPrice=df_month.iloc[df_month.index.searchsorted(predicted_date)]["Close"]
    currentPrice=df_month.iloc[df_month.index.searchsorted(current_date)]["Close"]

    interval = PREDICT_INTERVAL
    history_scores = []
    for x in reversed(range(interval-1,backward_list.shape[0])):
        c = backward_list.iloc[x-interval+1:x]
        del c["Date"]
        history_scores.append(model.score(c))
    history_scores = np.flipud(history_scores)

    most_likelihood_idx = np.argmin(np.absolute(history_scores - current_score))
#     most_likelihood_idx = most_likelihood_idx + interval
    predict_change = df_month.iloc[most_likelihood_idx+1]["Close"] - df_month.iloc[most_likelihood_idx]["Close"]
    predict_change = predict_change * np.sign(current_score - history_scores[np.argmin(np.absolute(history_scores -
current_score))])
    predictedPrice = currentPrice + predict_change


    most_likelihood_date = df_month.iloc[most_likelihood_idx]["Date"]
#     print(current_date, predicted_date, most_likelihood_date, currentPrice, actualPrice, predictedPrice, predict_change)

    return
currentPrice,actualPrice,predictedPrice,transmat_retune_prior,means_retune_prior,startprob_retune_prior,covars_retune_prior


# In[5]:


def run_predict():
    col_names =  ['predictedDate','currentDate', 'actualPrice', 'predictedPrice','currentPrice']
    df_pricePredicted  = pd.DataFrame(columns = col_names)

    start_date=''
    predicted_date=''
    current_date=''
    for month_count in range(0,PREDICT_INTERVAL):
        start_date = str(1996 + (month_count // 12 ))+'-'+str((month_count % 12) + 1) +'-01'
        current_date = str(2006 + ((month_count+1) // 12 ))+'-'+str(((month_count+1) % 12) + 1)+'-01'
        predicted_date = str(2006 + ((month_count+2) // 12 ))+'-'+str(((month_count+2) % 12) + 1 )+'-01'

        if month_count==0:

currentPrice,actualPrice,predictedPrice,transmat_retune_prior,means_retune_prior,startprob_retune_prior,covars_retune_prior=pre
dictPrice(df_month,start_date, predicted_date,current_date)
        else:

currentPrice,actualPrice,predictedPrice,transmat_retune_prior,means_retune_prior,startprob_retune_prior,covars_retune_prior=pre
dictPrice(df_month,start_date,
predicted_date,current_date,transmat_retune_prior,means_retune_prior,startprob_retune_prior,covars_retune_prior)
        df_pricePredicted.loc[month_count] = [predicted_date,current_date,actualPrice,predictedPrice,currentPrice]
    return df_pricePredicted


# In[6]:


def count_accuracy(df_pricePredicted):
    accuracy_count = 0
    sensitivity_count = 0
    specificity_count = 0

    actual_up_count = 0
    actual_down_count = 0

    for i in range(len(df_pricePredicted)):
        item_actualPrice = df_pricePredicted["actualPrice"][i]
        item_currentPrice = df_pricePredicted["currentPrice"][i]
```

```
        item_predictedPrice = df_pricePredicted["predictedPrice"][i]

        item_actualChange = item_actualPrice - item_currentPrice
        item_predictChange = item_predictedPrice - item_currentPrice

        if item_actualChange * item_predictChange > 0:
            accuracy_count += 1
        if item_actualChange > 0 and item_predictChange > 0:
            sensitivity_count += 1
        if item_actualChange < 0 and item_predictChange < 0:
            specificity_count += 1

        if item_actualChange > 0:
            actual_up_count += 1
        else:
            actual_down_count += 1

    print("accuracy:", accuracy_count/len(df_pricePredicted))
    print("sensitivity:", sensitivity_count/actual_up_count)
    print("specificity:", specificity_count/actual_down_count)


# In[7]:


PREDICT_INTERVAL = 120


# In[8]:


result_df = run_predict()
count_accuracy(result_df)


# In[9]:


PREDICT_INTERVAL = 80


# In[10]:


result_df = run_predict()
count_accuracy(result_df)


# In[11]:


PREDICT_INTERVAL = 40


# In[12]:


result_df = run_predict()
count_accuracy(result_df)


# In[13]:


# (result_df["currentPrice"] == result_df["predictedPrice"]).any()

#plot chart of Comparing actual price with price predicted from Gaussian HMM

def showData2(array_nodate, title, transparency=1.0):
    df_nodate_prep={'Actual':df_pricePredicted['actualPrice'], 'Predicted':df_pricePredicted['predictedPrice']}
```

```
    df_nodate=pd.DataFrame(data=df_nodate_prep)
    s=df_nodate
    fig, ax = plt.subplots()
    ax.plot(s,alpha=transparency)
    plt.legend(["actual","predicted"])
    plt.title(title)
    plt.xlabel('months (from 2006-3 to 2016-2-01)')
    plt.ylabel('close price')
showData2(df_pricePredicted,"hmm")
```