DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Homework Assignment No. 06:

# HW No. 06: Expectation Maximization (EM) and Hidden Markov Models (HMMs)

submitted to:

Professor Joseph Picone
ECE 8527: Introduction to Pattern Recognition and Machine Learning
Temple University
College of Engineering
1947 North 12th Street
Philadelphia, Pennsylvania 19122

February 9th, 2023

prepared by:

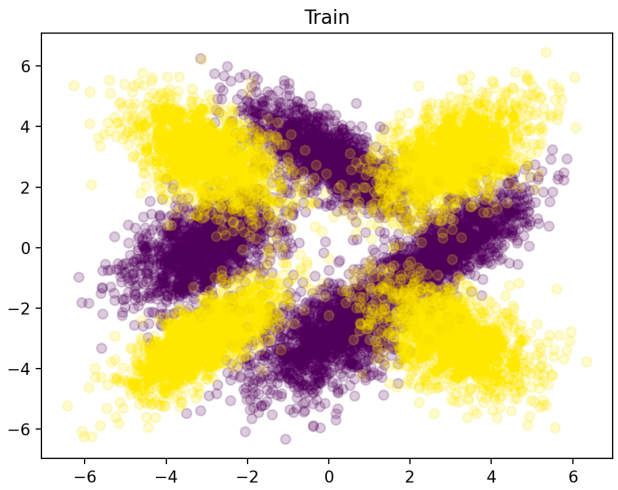
Gavin Koma // Joe Arthur
Email: gavintkoma@temple.edu // joe.arthur@temple.edu

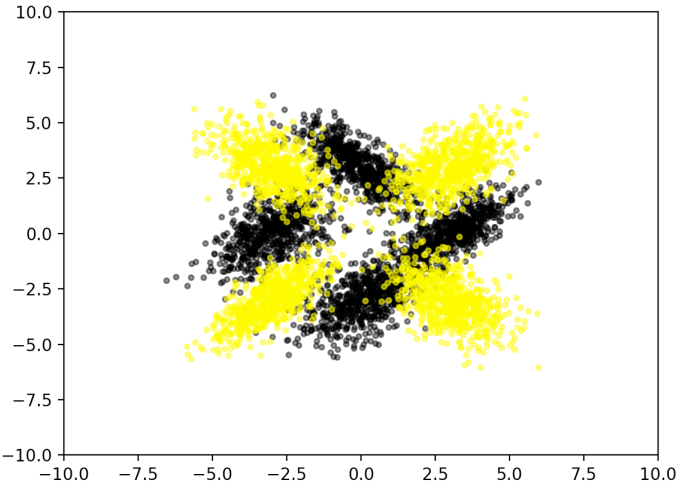
## A. TASK 1

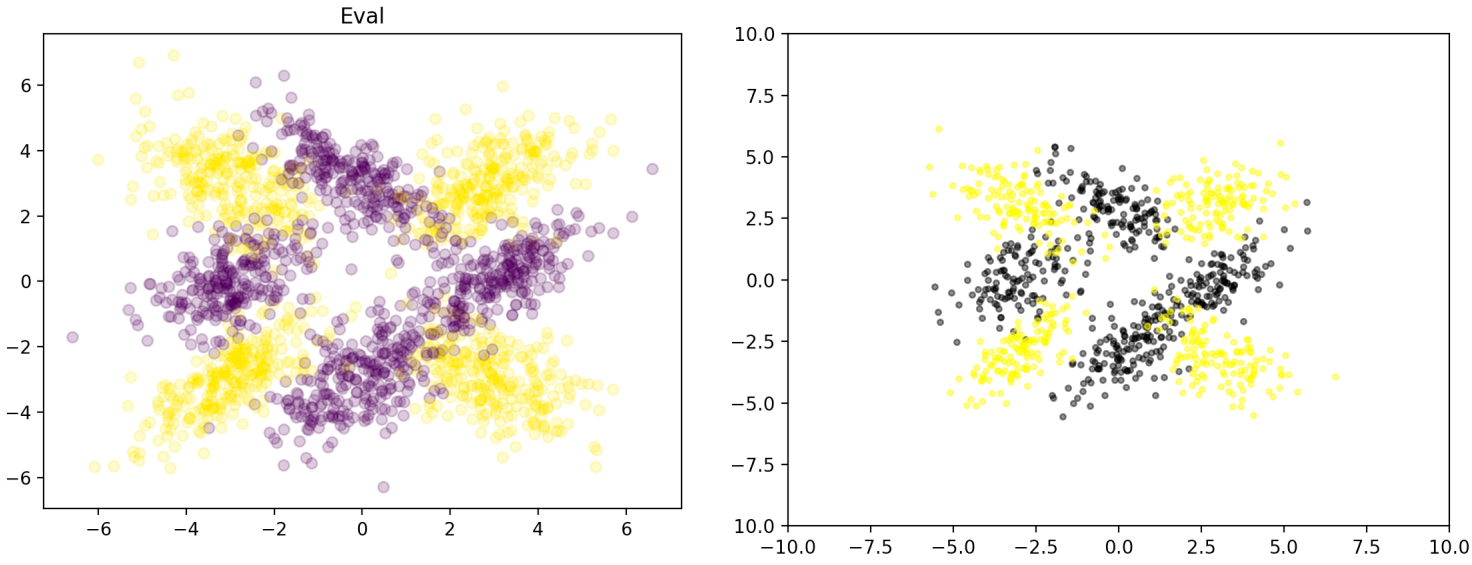
The goal of the first task was to model each of the given classes as a mixture of multivariate Gaussian distributions and then to estimate the parameters of these distributions using EM. In order to begin, I first plotted the data to ensure that it was similar to the provided graphs. In order to do so, I wrote a simple function which passed the varying datasets to a plot. The code is as follows:

```
#%%lets begin
#change directory to data (2d)
os.chdir(r'/Users/gavinkoma/Desktop/pattern_rec/homework6/2d/data')
print(os.getcwd())

#import the data bruv
filetrain = np.loadtxt("train.txt", dtype=float)
y_train = pd.DataFrame(filetrain[:,0])
x_train = pd.DataFrame(filetrain[:,1:3])

fileeval = np.loadtxt("eval.txt", dtype=float)
y_eval = pd.DataFrame(fileeval[:,0])
x_eval = pd.DataFrame(fileeval[:,1:3])

name_dict = {0:"Train",
             1:"Eval"
             }

def plot(file):
    plt.figure()
    #print(kval[1],kval[2])

    plt.scatter(file[:,1],file[:,2],c=file[:,0],alpha=0.2)
    plt.title(name_dict[count])

for count in [0]:
    plot(filetrain)
    for count in [1]:
        plot(fileeval)
```

I have included my outputted graphs and attached them below in order to compare them to the provided graphs. One can see that they look the same. From here, I am comfortable with moving forward.



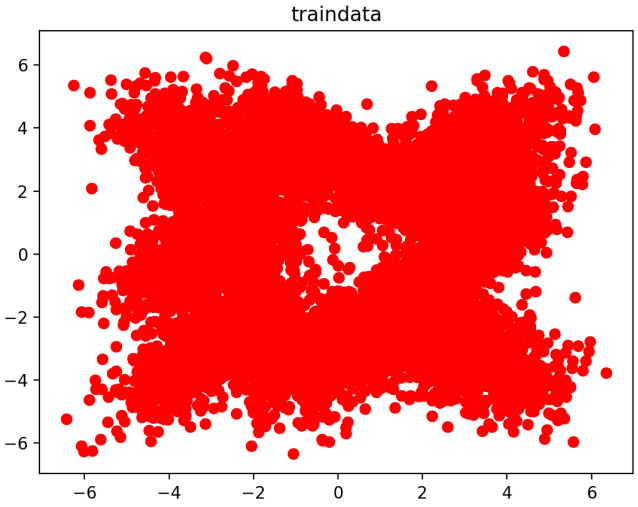


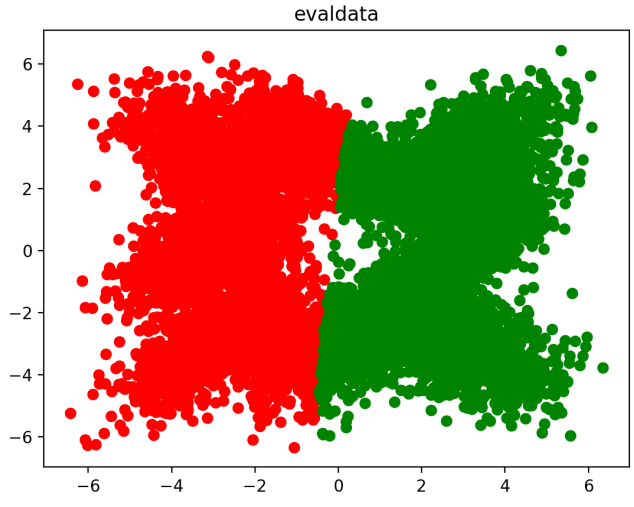

The implementation of Gaussian Mixture Models using python is fairly simple and only requires a few lines of code. However, we also need to show the boundaries that are estimated by this model. To do so, I implemented a similar strategy that I used when making the previous two plots. There are two main goals of using the Gaussian Mixture Model. First, to estimate the parameters of each Gaussian component that is within the Gaussian Mixture. The second is to determine which Gaussian component a data point will belong to.

```
#%%
#there are two different classes here 0,1 --> binary --> love!
#why dont we plot first to get an idea of stuff

def GM_plot(x,y):
    plt.figure()
    d = x
    gmm.fit(d)
    labels = gmm.predict(d)
    d['labels'] = labels
    d0 = d[d['labels'] == 0]
    d1 = d[d['labels'] == 1]
    plt.scatter(d0[0],d0[1],c='r')
    plt.scatter(d1[0],d1[1],c='g')
    plt.title(name_dict[i])
    plt.show()

name_dict = {
    0: "traindata",#data10train
    1: "evaldata", #data9train
    2: "traindata", #data8train
    3: "evaldata", #data9eval
    }

for i,val in enumerate([1,2,4,8]):
    gmm = GaussianMixture(n_components=val,random_state=0)
    GM_plot(x_train,y_train)

for i,val in enumerate([1,2,4,8]):
    gmm = GaussianMixture(n_components=val,random_state=0)
    GM_plot(x_eval,y_eval)
```

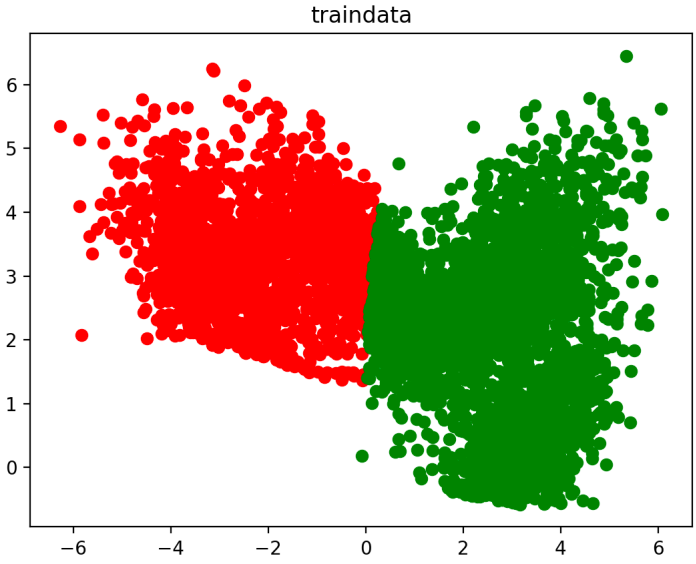
The implementation of the code was incredibly simple. I passed the same previous files that were used to plot normally and instead passed them through a Gaussian Mixture Estimator. Doing so provided me with the following graphs for varying values of n_components.

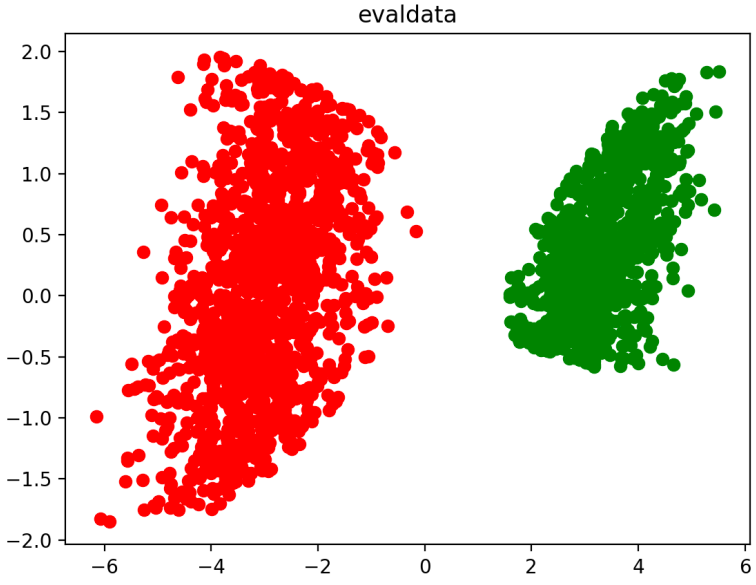
For n_components = 1:



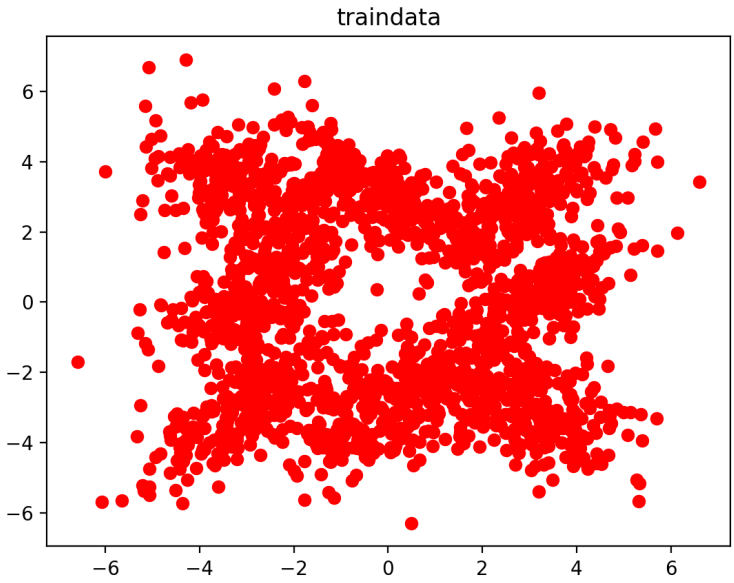



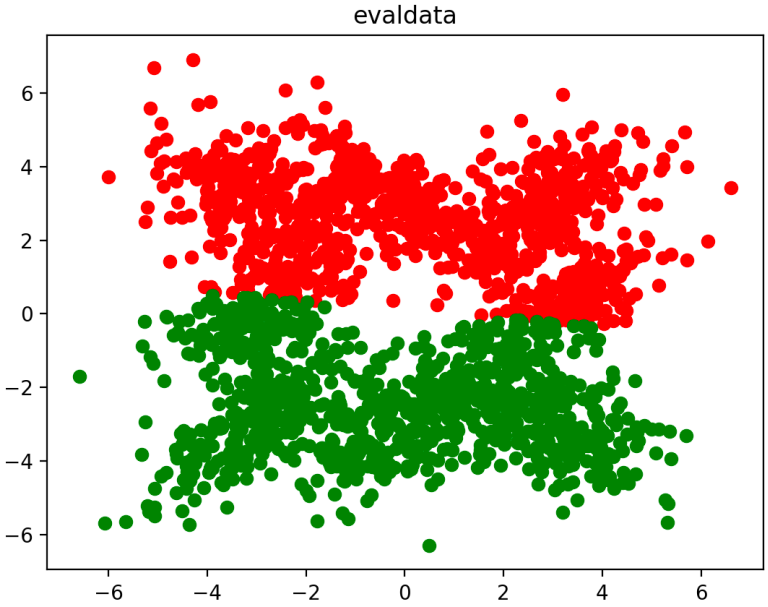

For n_components = 2:



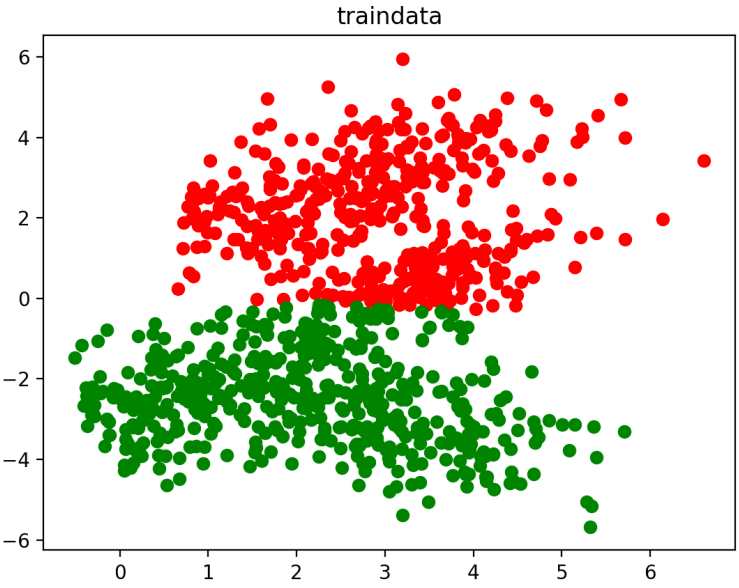



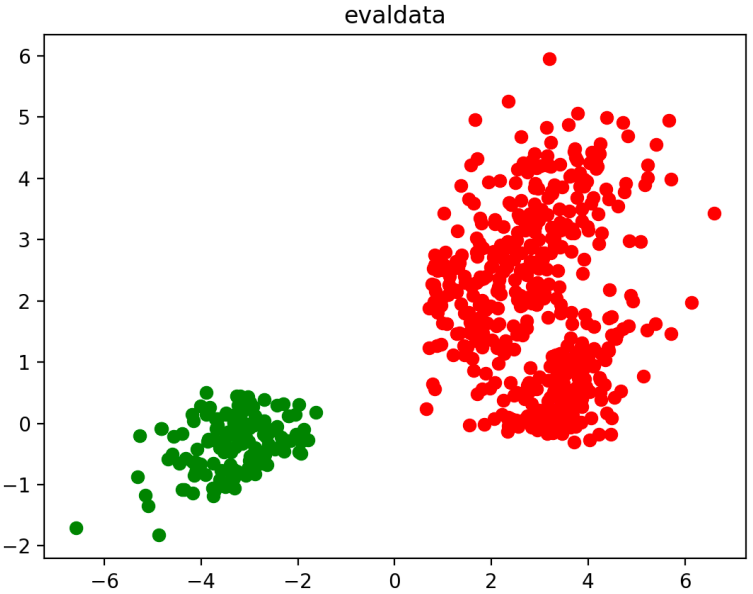
For n_components = 4:




For n_components = 8:

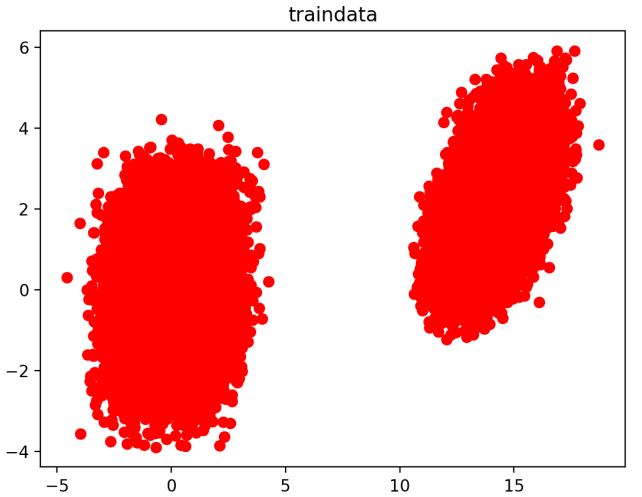


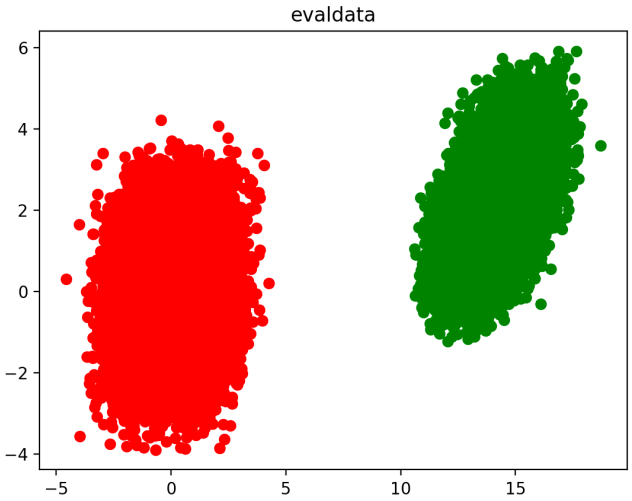



## B. TASK 2

The goal of Task 2 was to implement the same Gaussian Mixture method as previously, but with data that has more than two features. In this example, we are given a dataset with 5 features (making it a little hard to plot without some type of feature transformation). I went through this dataset in the same manner that I had previously. I used a for-loop that allowed me to train a model, pass the data, and output a corresponding plot. However, I felt that this was not entirely descriptive as the data points overlay on top of each other. The code that was used to implement this technique on the 5d data is as follows:

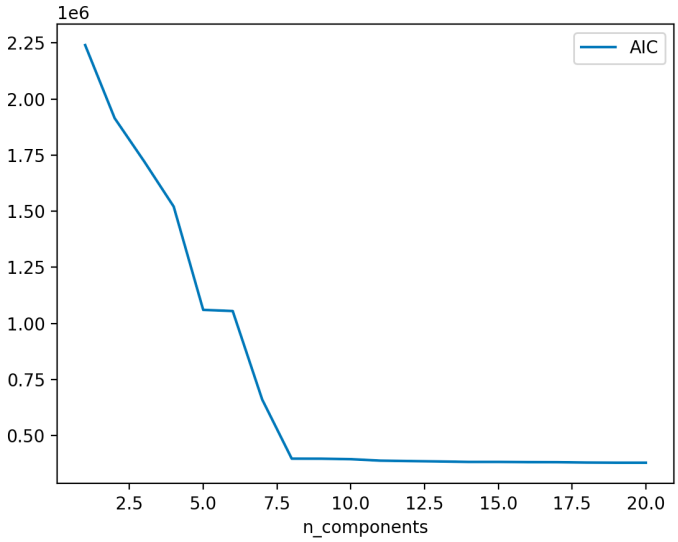
```
#%% get 5d data
os.chdir(r'/Users/gavinkoma/Desktop/pattern_rec/homework6/5d')
print(os.getcwd())

filetrain = np.loadtxt('train.txt',dtype=float)
y_train = pd.DataFrame(filetrain[:,0])
x_train = pd.DataFrame(filetrain[:,1:6])

fileeval = np.loadtxt('eval_anonymized.txt',dtype=float)
y_eval = pd.DataFrame(fileeval[:,0])
x_eval = pd.DataFrame(fileeval[:,1:6])

name_dict = {0:'train_5d',
             1:'eval_5d'
             }

#cant really plot this? so i think im just gonna keep going

def GM_plot(x,y):
    plt.figure()
    d = x
    gmm.fit(d)
    labels = gmm.predict(d)
    d['labels'] = labels
    d0 = d[d['labels'] == 0]
    d1 = d[d['labels'] == 1]
    plt.scatter(d0[0],d0[1],c='r')
    plt.scatter(d1[0],d1[1],c='g')
    plt.title(name_dict[i])
    plt.show()

name_dict = {
    0: "traindata",#data10train
    1: "evaldata", #data9train
    2: "traindata", #data8train
    3: "evaldata", #data9eval
    }

for i,val in enumerate([1,2,4,8]):
    gmm = GaussianMixture(n_components=val,random_state=0)
    GM_plot(x_train,y_train)

for i,val in enumerate([1,2,4,8]):
    gmm = GaussianMixture(n_components=val,random_state=0)
    GM_plot(x_eval,y_eval)
```




I realized quickly that viewing the boundaries with higher dimensional data would not be a satisfactory option. I did a bit more reading and found another method to assess n_components and how to 'judge' our model. One such analytic technique is the Akaike Information Criterion (AIC). This method allows us to estimate the prediction error and thereby assess the quality of our model given our data. This is fairly easy to implement in python and only required a small bit of list comprehension to loop through our data. The code is as such and the graph that is outputted is attached below:

```
#%%
#gmm = GaussianMixture(n_components=val,random_state=0)
from sklearn.mixture import GaussianMixture as GMM


n_components = np.arange(1, 21)
models = [GMM(n, covariance_type='full', random_state=0).fit(x_train) for n in n_components]

plt.plot(n_components, [m.aic(x_train) for m in models], label='AIC')
plt.legend(loc='best')
plt.xlabel('n_components')
```




This graph gives us a good idea of *how many* n_components we should include in our Gaussian Mixture to minimize probability of error. Here we see that at n_components of about 8 is probably satisfactory and anymore will not necessarily aid in the prediction rate of our model.

## C. TASK 3

Hidden Markov Models (HMMs) are used to model hidden Markov Processes. They are generally defined by 3 parameters:

1. Initial state probabilities. A vector that describes the initial probabilities of the system when it is in a particular state.
2. Hidden state transition matrix. The rows of this matrix will correspond to one particular hidden state while the columns for each of the row will contain transition probabilities from one state to a new hidden state.
3. Observable emission probabilities. This is a vector that will describe the emission probabilities for a process given a state.

The article continues to explain HMMs with a simple example regarding moods. It really simplified HMMs in my mind and helped me to gain a better understanding of them. I will continue to summarize this article in the following lines while including lines of my code to not only further my understanding but to also provide myself with materials for studying with at a later date.

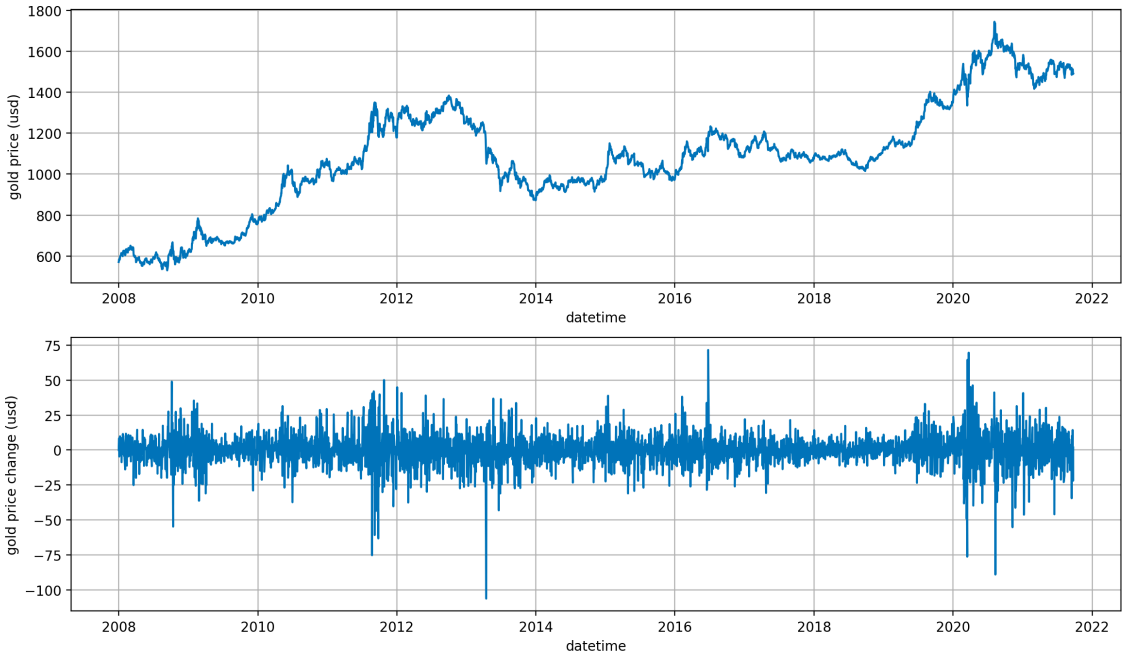
The article states that there are three general problems which are typically solved using hidden Markov

Models.

1. If we are given a set of observations X and the 3 model parameters, then we should calculate the occurrence probability of the observations of X. We can do this by using a forward algorithm.
2. If we are given a set of observations X and the three model parameters and we want to determine the optimal set of hidden states Z that result in X, then we should use the Viterbi algorithm.
3. Finally, if we are given only a set of observations X and our goal is to determine the optimal set of model parameters. Then we can use the Baum-Welch algorithm to solve this.

To follow along with this blog post, the code was followed word for word and graphs were outputted as the author did as well. The graphs are slightly different in color scheme but show the same data. I am slightly curious as to variations in number output, but I wonder if any data has changed from the git database.

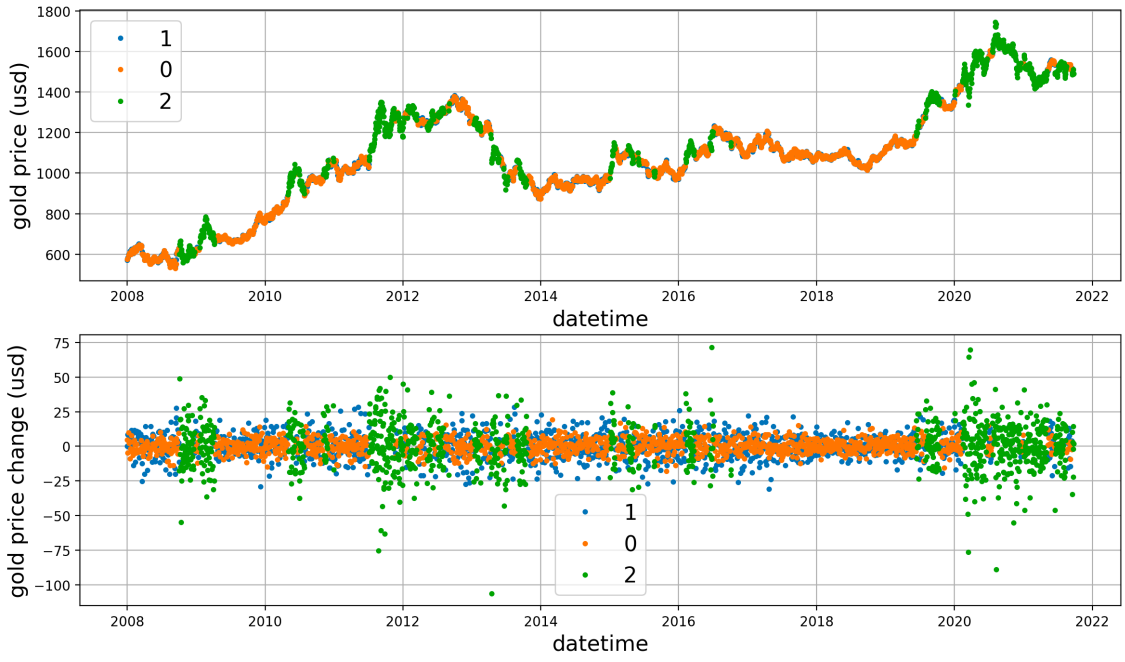
Our first graph shows and identical output and depicts the historic gold prices in USD as well as the change in the corresponding day change. We are better able model the daily change in the gold price and thereby better capture the changing state of the market. One important thing to note from this article is the reason for choosing 3 hidden states: we expect **at the very least** for there to be three different measure of daily changes (low, medium, and high).




We are able to confirm our states by viewing the output below. Here we see three states, given in no particular order. Another important thing to note is found in our transition matrix. The values in the diagonal are considered large as opposed to the smaller values that are outside of the diagonal. This allows us to know that our model prefers to remain in the state that it is already in.

```
In [63]: runcell('fit the daily change to ge model with 3 hidden states', '/
Users/gavinkoma/Desktop/pattern_rec/homework6/hmm.py')
Unique states:
[1 0 2]

Start probabilities:
[9.00876474e-03 9.90991235e-01 1.90652535e-52]

Transition matrix:
[[4.71987506e-02 9.52205396e-01 5.95853639e-04]
 [8.12868067e-01 1.35345228e-01 5.17867054e-02]
 [3.95757463e-02 4.27802116e-02 9.17644042e-01]]

Gaussian distribution means:
[[0.27897404]
 [0.20658205]
 [0.30620104]]

Gaussian distribution covariances:
[[[ 28.11246244]]

 [[ 77.18151421]]

 [[324.43713708]]]
```

The final portion of this task required us to output an image that shows market volatility when it has been modeled using a Gaussian emissions Hidden Markov Model. Our model varies slightly in that our state 1 is blue and state 0 is orange while the author of this paper has state 1 as orange and state 0 as blue. This causes it to look slightly different but the outputted states from previous sections show that our model would most likely start in state 1.



An image of the short code was included below:

```
#%% get data
#dont need to change the wd for this one! woo!

base_dir = "https://github.com/natsunoyuki/Data_Science/blob/master/gold/gold/gold_pric

data = pd.read_csv(base_dir)
data["datetime"] = pd.to_datetime(data["datetime"])
data["gold_price_change"] = data["gold_price_usd"].diff()
data = data[data["datetime"] >= pd.to_datetime("2008-01-01")]

plt.figure(figsize = (15, 10))
plt.subplot(2,1,1)
plt.plot(data["datetime"], data["gold_price_usd"])
plt.xlabel("datetime")
plt.ylabel("gold price (usd)")
plt.grid(True)
plt.subplot(2,1,2)
plt.plot(data["datetime"], data["gold_price_change"])
plt.xlabel("datetime")
plt.ylabel("gold price change (usd)")
plt.grid(True)
plt.show()
```

```
#%%fit the daily change to ge model with 3 hidden states
X = data[["gold_price_change"]].values
model = hmm.GaussianHMM(n_components = 3, covariance_type = "diag",
                        n_iter = 50, random_state = 42)
model.fit(X)

Z = model.predict(X)
states = pd.unique(Z)

print("Unique states:")
print(states)

print("\nStart probabilities:")
print(model.startprob_) #highest prob is found in state 0

print("\nTransition matrix:")
print(model.transmat_)

print("\nGaussian distribution means:")
print(model.means_)

print("\nGaussian distribution covariances:")
print(model.covars_)
```

```
#%% and now we want to plot
plt.figure(figsize = (15, 10))
plt.subplot(2,1,1)
for i in states:
    want = (Z == i)
    x = data["datetime"].iloc[want]
    y = data["gold_price_usd"].iloc[want]
    plt.plot(x, y, '.')
plt.legend(states, fontsize=16)
plt.grid(True)
plt.xlabel("datetime", fontsize=16)
plt.ylabel("gold price (usd)", fontsize=16)
plt.subplot(2,1,2)
for i in states:
    want = (Z == i)
    x = data["datetime"].iloc[want]
    y = data["gold_price_change"].iloc[want]
    plt.plot(x, y, '.')
plt.legend(states, fontsize=16)
plt.grid(True)
plt.xlabel("datetime", fontsize=16)
plt.ylabel("gold price change (usd)", fontsize=16)
plt.show()
```

## D. CONCLUSION

I genuinely enjoyed this homework assignment. A lot of the work that we've done so far has always included applications of the algorithms of some manner and it's been really helpful for me to be able to visualize these ideas. I think the difference for this assignment was the article that was attached and offered further explanation while looking at the code. Normally, I have to deduce the assumptions included in a library or figure it out through trial and error, but it was really nice to have the code and explanation next to each other. I think it definitely helped to facilitate my understanding of *how* as well as *why* the Gaussian Mixture Model works the way it does.