Assignment 4

Hidden Markov Models:

Preparing the data:

The following code was used:

```
[9] # Only taking Centroid coordinates (X and Y) for Left and Right hands
    # LC X,Y
    lc = data.iloc[:,:60].to_numpy()
    print(lc.shape)

    # RC X,Y
    rc = data.iloc[:,390:450].to_numpy()
    print(rc.shape)

    # reshape
    new_LC = lc.reshape(350,30,2)
    new_RC = rc.reshape(350,30,2)
    print(new_LC.shape)
    print(new_RC.shape)

    # final data
    new_data = np.concatenate((new_LC, new_RC), axis=2)
    print(new_data.shape)
    #print(new_data)

    (350, 60)
    (350, 60)
    (350, 30, 2)
    (350, 30, 2)
    (350, 30, 4)
```

Normalizing the data:

The following code was used to normalize:

```
[10] #Define normalize function
     def normalize(data):
         min_data = (data - np.min(data))
         max_data = (np.max(data) - np.min(data))
         normalized_data = min_data / max_data
         return normalized_data

[11] #Applying function to dataset
     data = normalize(new_data)
```

Building and training the models:

7 models were created and trained as same as shown below:

Training the model

```
[17] # Model 1 training
     cls1 = GaussianHMM()
     cls1.fit(m1_train.reshape(-1, 4))

     GaussianHMM()

[18] # Model 2 training
     cls2 = GaussianHMM()
     cls2.fit(m2_train.reshape(-1, 4))

     GaussianHMM()

[19] # Model 3 training
     cls3 = GaussianHMM()
     cls3.fit(m3_train.reshape(-1, 4))

     GaussianHMM()

[20] # Model 4 training
     cls4 = GaussianHMM()
     cls4.fit(m4_train.reshape(-1, 4))
```

Evaluating test data:

Below are the results showing likelihood and most likelihoods generated by applying trained models on testing set.

For every result first line contains list of likelihoods and second line shows the maximum likelihood:

```
[47.4558, -46.3687, -1324406.8118, -1324395.4743, -1324394.2235, -1324393.473, -1324397.7309]
0

[61.3487, 6.9248, -1170704.5748, -1170676.9641, -1170685.2713, -1170684.3005, -1170693.801]
0

[47.0138, -67.7304, -1380664.7488, -1380658.09, -1380655.8653, -1380643.5807, -1380647.5062]
0

[98.1558, 96.2088, -1188028.1611, -1187979.7587, -1188000.214, -1187999.2802, -1188015.572]
0

[89.4107, 83.4099, -1437780.4318, -1437733.7823, -1437753.7396, -1437749.27, -1437765.331]
0

[96.5861, 119.2816, -1314930.1623, -1314877.2288, -1314898.0827, -1314911.84, -1314928.0753]
1

[92.9787, 63.6454, -1386084.9803, -1386043.9914, -1386061.8844, -1386049.2325, -1386064.3012]
0

[75.173, 2.2566, -1530221.4492, -1530196.0778, -1530205.9559, -1530185.965, -1530196.7384]
0

[54.5207, 48.0442, -1344256.163, -1344218.0947, -1344229.6784, -1344247.1307, -1344258.2035]
0
```

Below is the accuracy of these predictions against the actual test data: 72.86%

```
[30] acc = accuracy_score(predictions, actual_test )
     print("Accuracy is ", acc*100)

     Accuracy is  72.85714285714285
```

Finding the best configuration:

Definition of Component 1:

```
#First try of hyperparameter search by training the models us

comp1_model1 = GaussianHMM(n_components=12)
comp1_model2 = GaussianHMM(n_components=14)
comp1_model3 = GaussianHMM(n_components=10)
comp1_model4 = GaussianHMM(n_components=17)
comp1_model5 = GaussianHMM(n_components=13)
comp1_model6 = GaussianHMM(n_components=15)
comp1_model7 = GaussianHMM(n_components=19)
```

Accuracy of Component 1:

```
#Component 1 accuracy
comp1_acc = accuracy_score(comp1_predictions, actual_test )
print("Accuracy is ", comp1_acc*100)

Accuracy is  82.85714285714286
```

Definition of Component 2:

```
[36] #Second try of hyperparameter search by training the models us

     comp2_model1 = GaussianHMM(n_components=2)
     comp2_model2 = GaussianHMM(n_components=4)
     comp2_model3 = GaussianHMM(n_components=8)
     comp2_model4 = GaussianHMM(n_components=6)
     comp2_model5 = GaussianHMM(n_components=10)
     comp2_model6 = GaussianHMM(n_components=7)
     comp2_model7 = GaussianHMM(n_components=5)
```

Accuracy of Component 2:

```
[40] #Component 2 accuracy
     comp2_acc = accuracy_score(comp2_predictions, actual_test )
     print("Accuracy is ", comp2_acc*100)

     Accuracy is  81.42857142857143
```

Definition of Component 3:

```
[41] #Third try of hyperparameter search by training the m

     comp3_model1 = GaussianHMM(n_components=29)
     comp3_model2 = GaussianHMM(n_components=21)
     comp3_model3 = GaussianHMM(n_components=22)
     comp3_model4 = GaussianHMM(n_components=29)
     comp3_model5 = GaussianHMM(n_components=27)
     comp3_model6 = GaussianHMM(n_components=27)
     comp3_model7 = GaussianHMM(n_components=26)
```

Accuracy of Component 3:

```
[45] #Component 3 accuracy
     comp3_acc = accuracy_score(comp3_predictions, actual_test )
     print("Accuracy is ", comp3_acc*100)

     Accuracy is  87.14285714285714
```

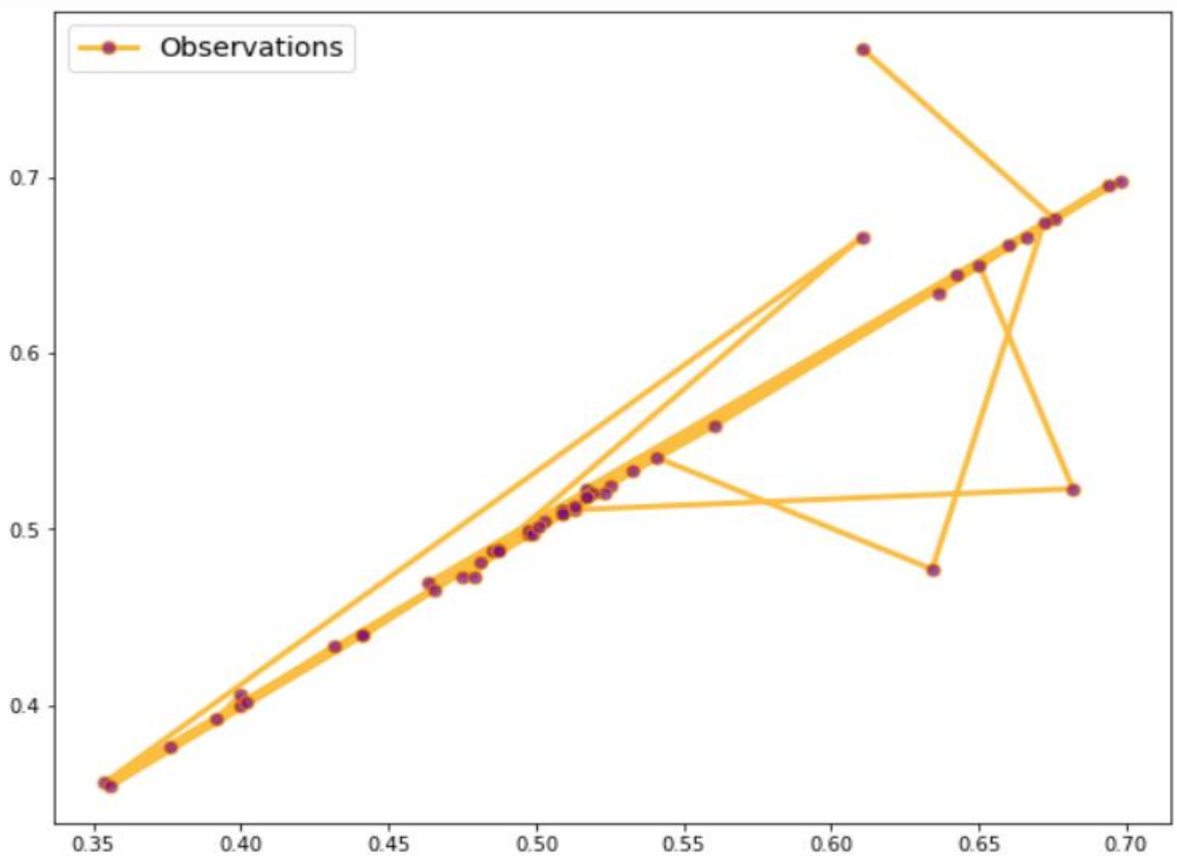
Comparison Table:

Component 1	Component 2	Component 3
82.86%	81.43%	**87.14%**

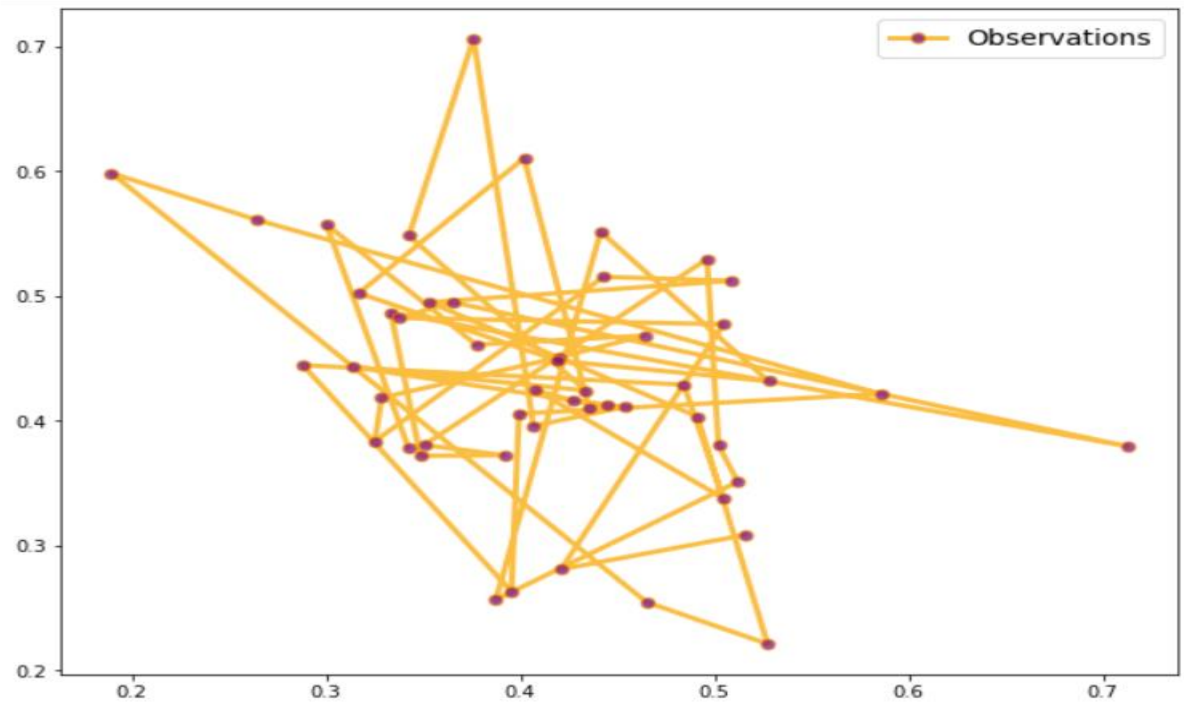
Sampling from HMM:

Model 1:

Training:

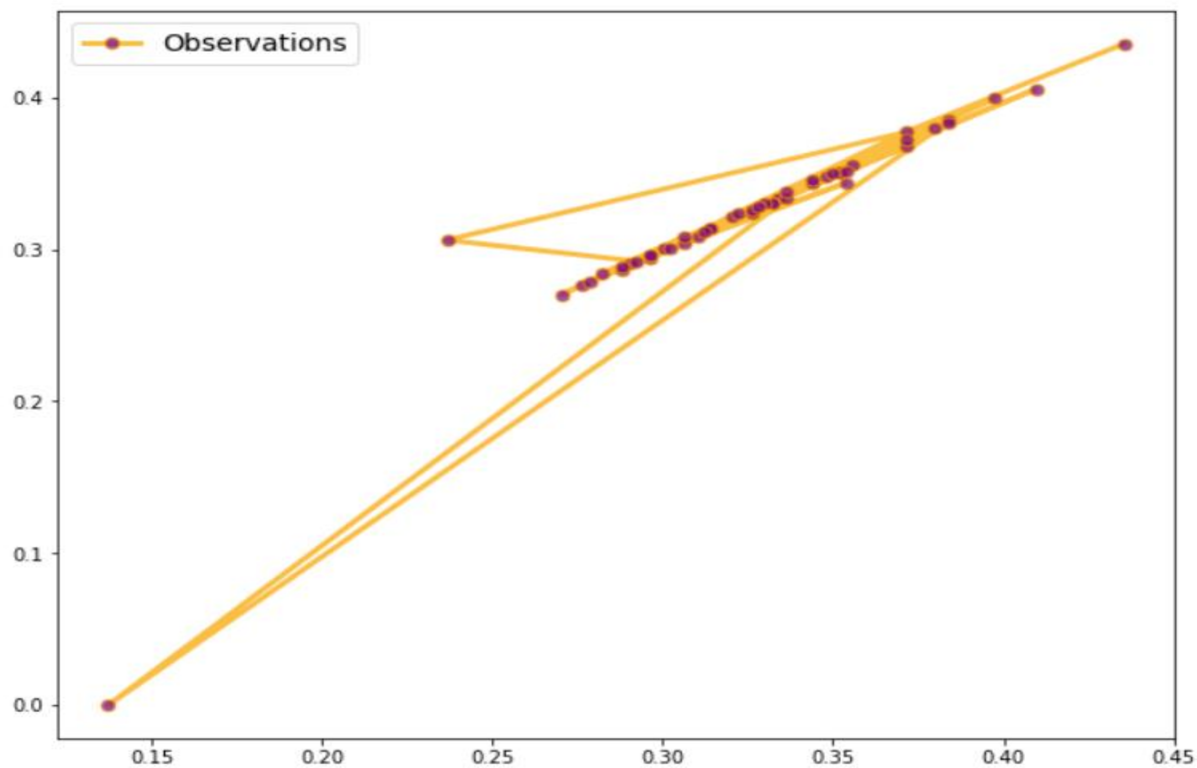


Testing (Generated):

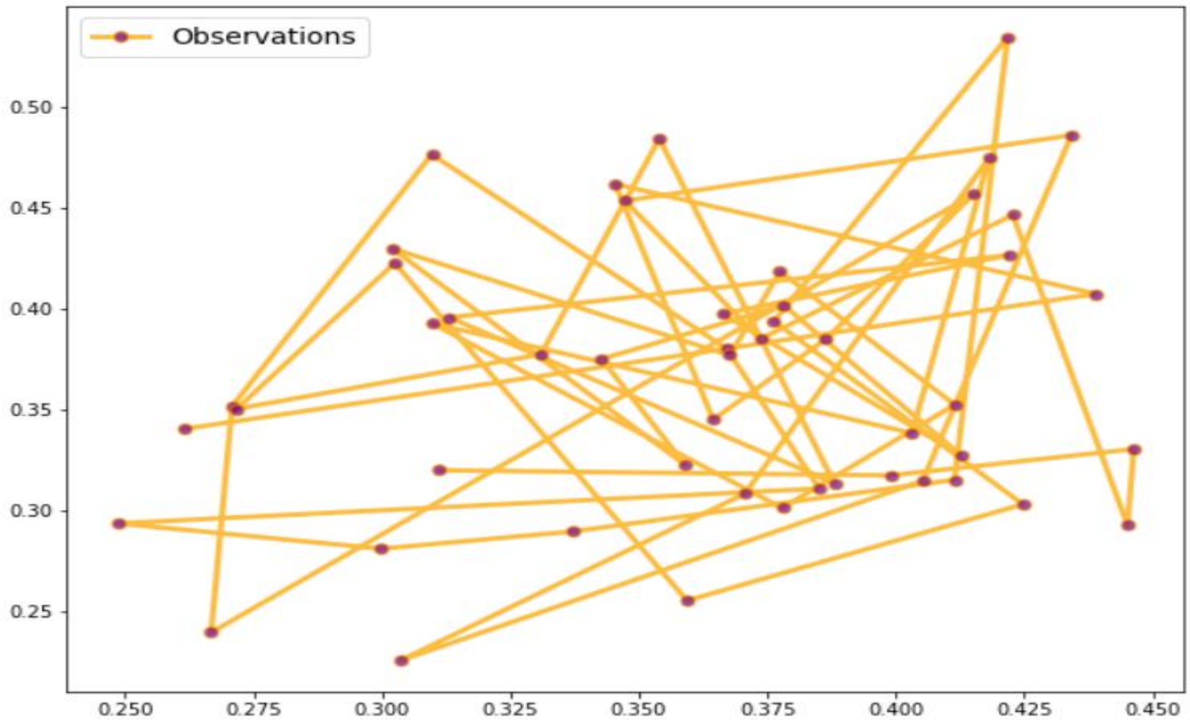


Model 2:

Training:

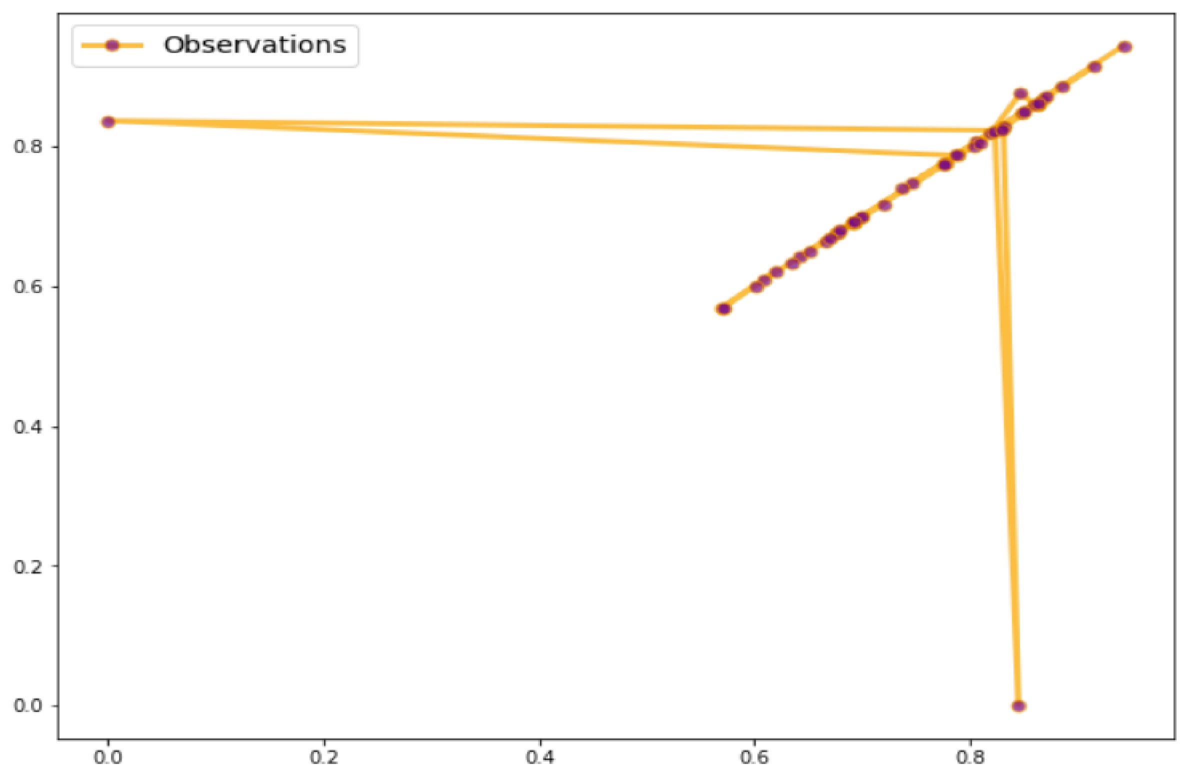


Testing (Generated):

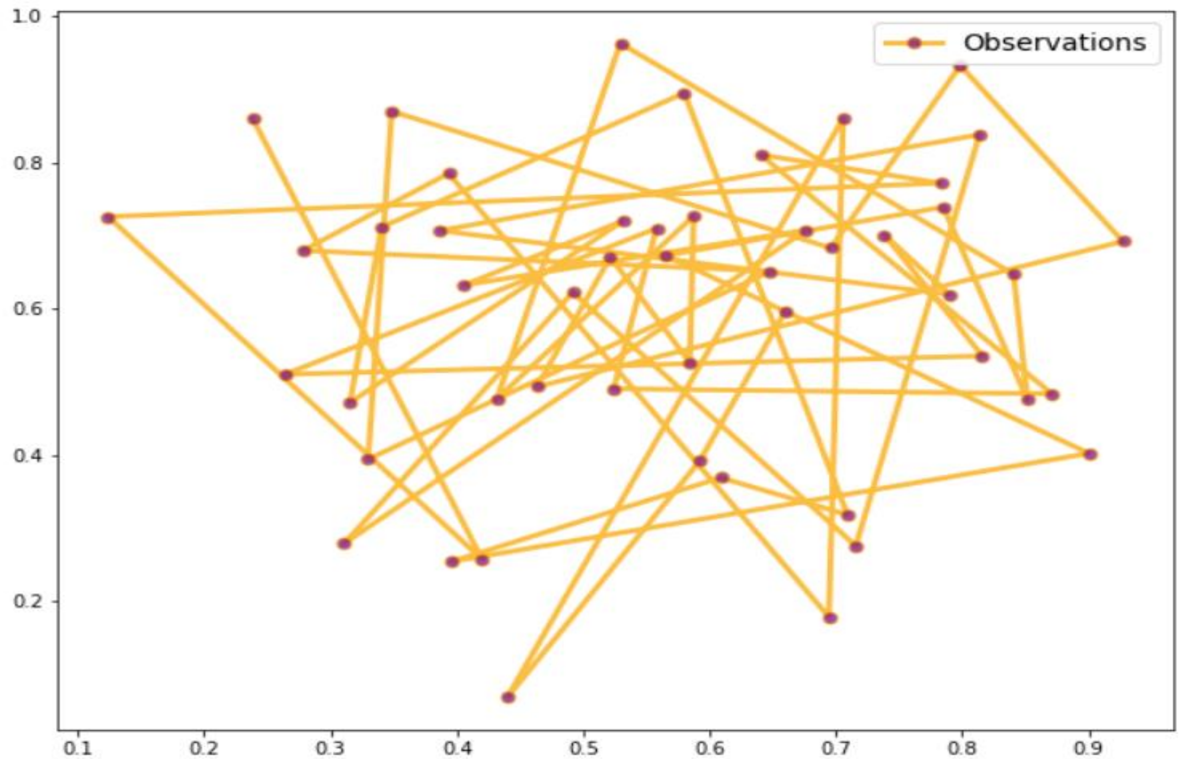


Model 3:

Training:

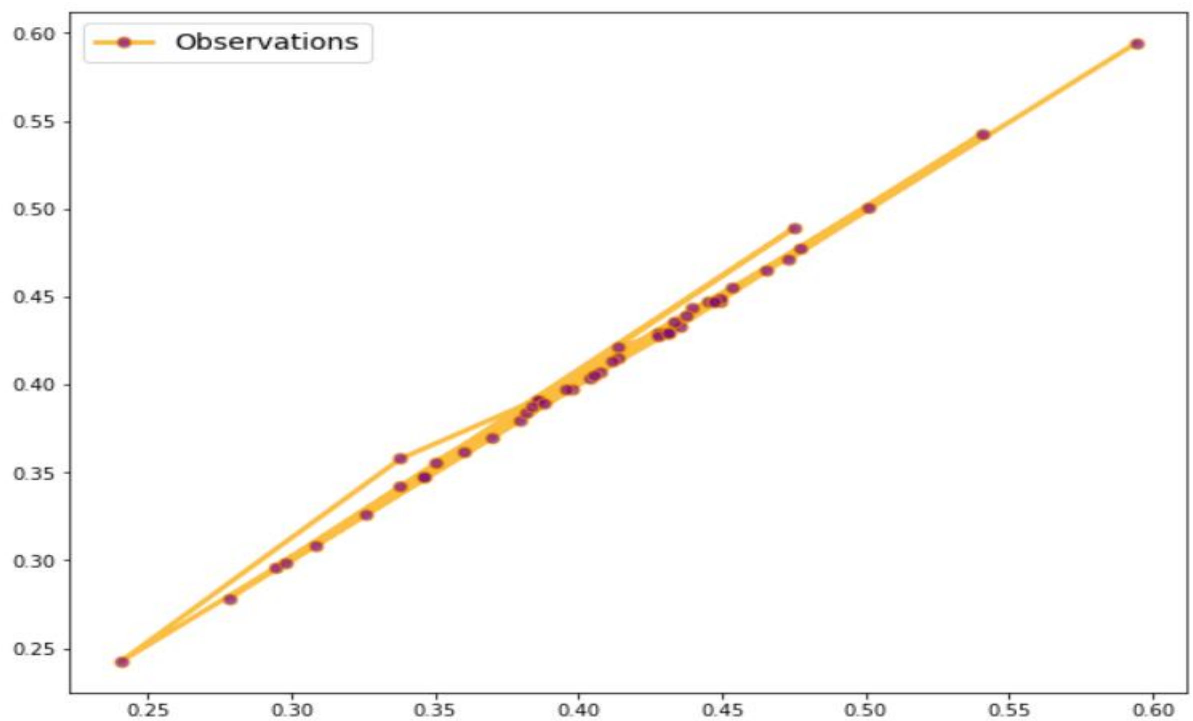


Testing (Generated):

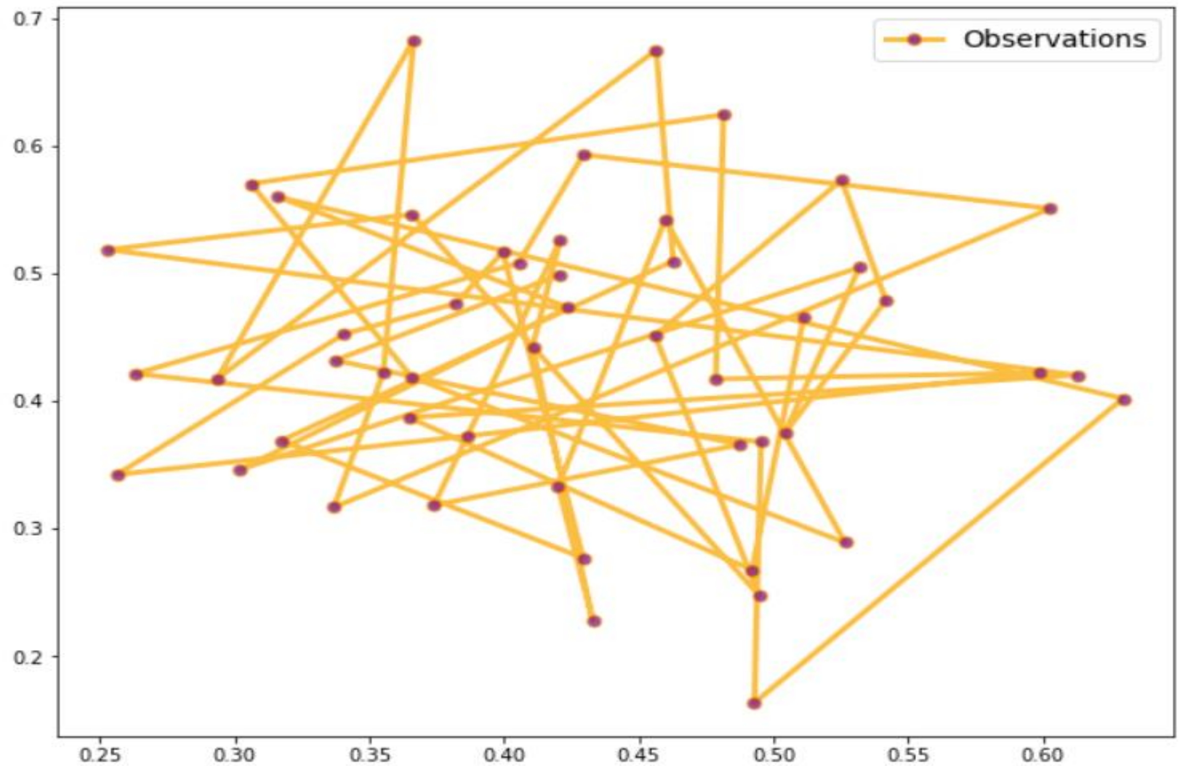


Model 4:

Training:

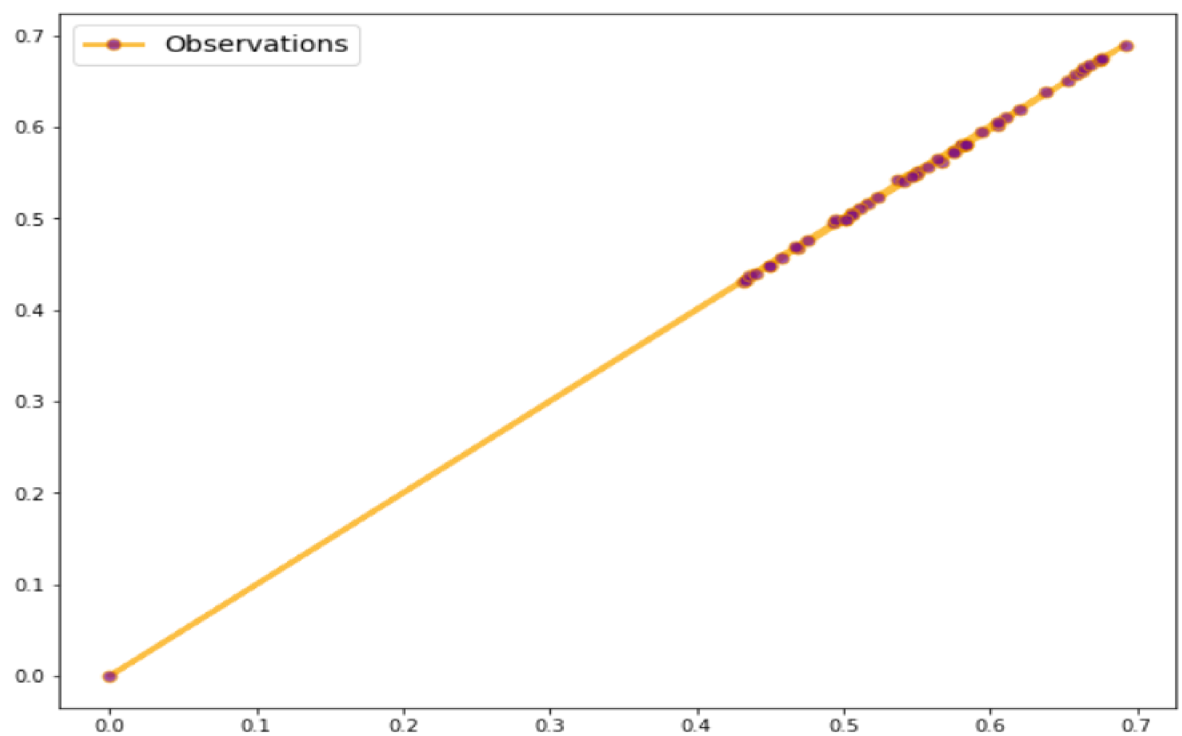


Testing (Generated):

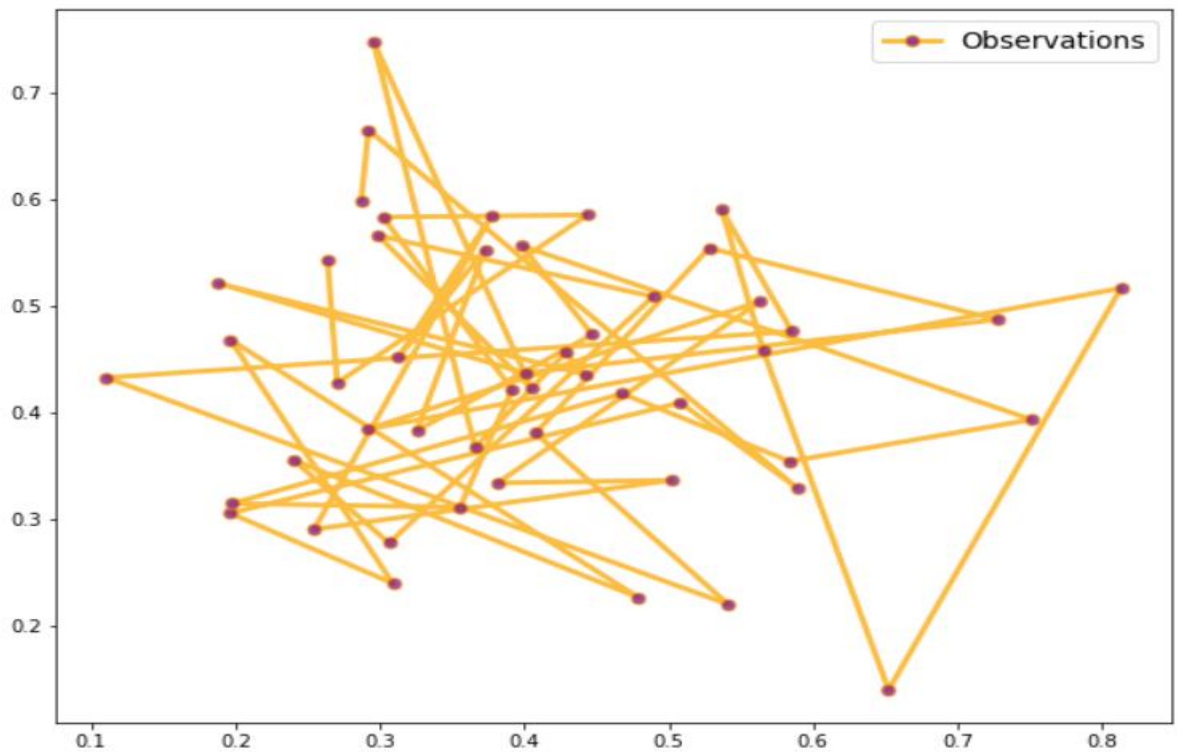


Model 5:

Training:

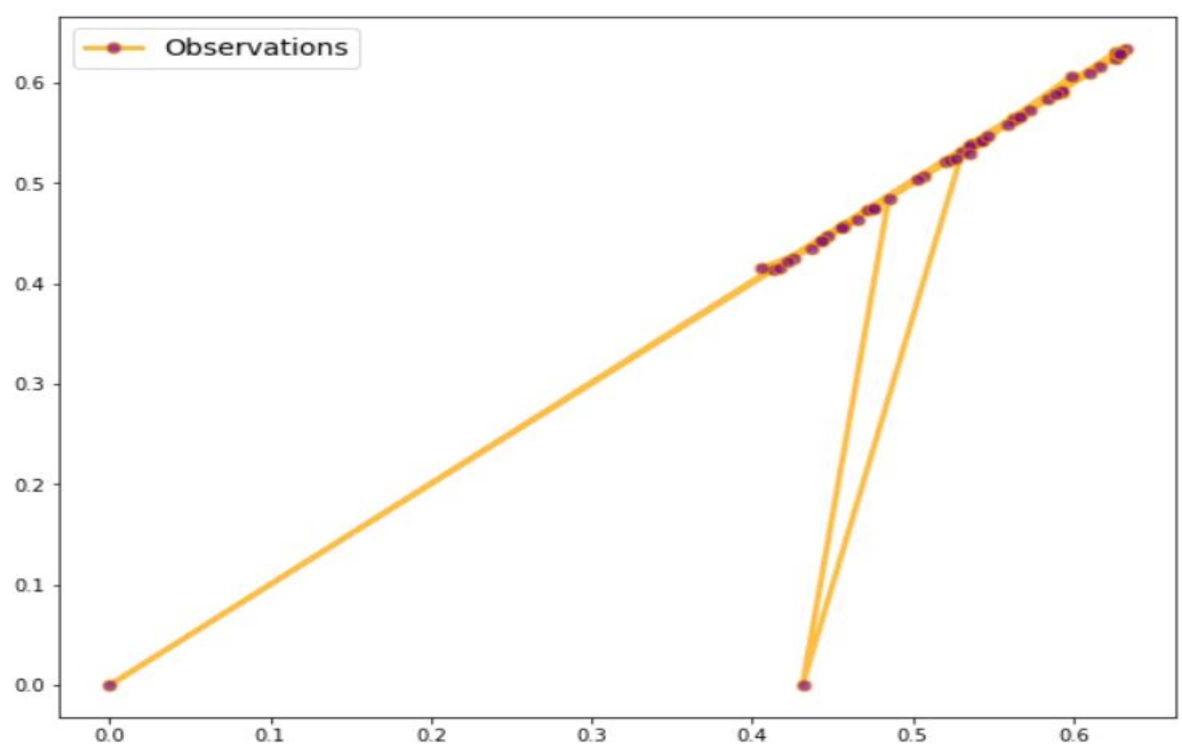


Testing (Generated):

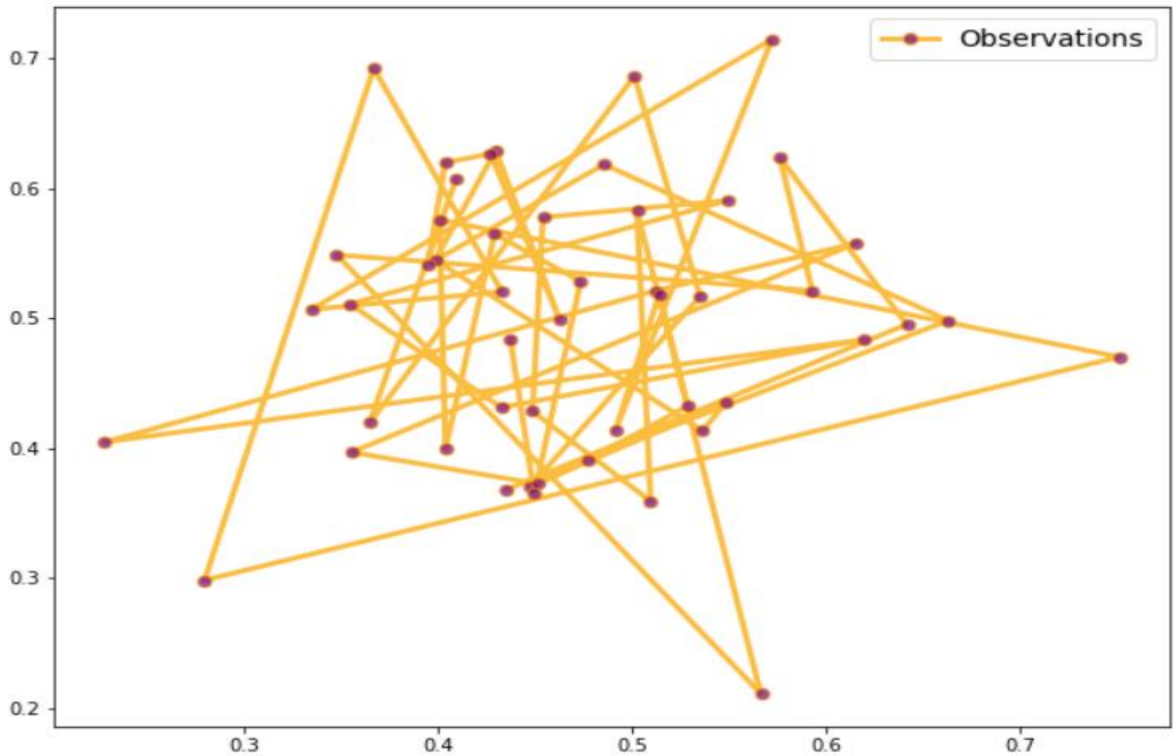


Model 6:

Training:

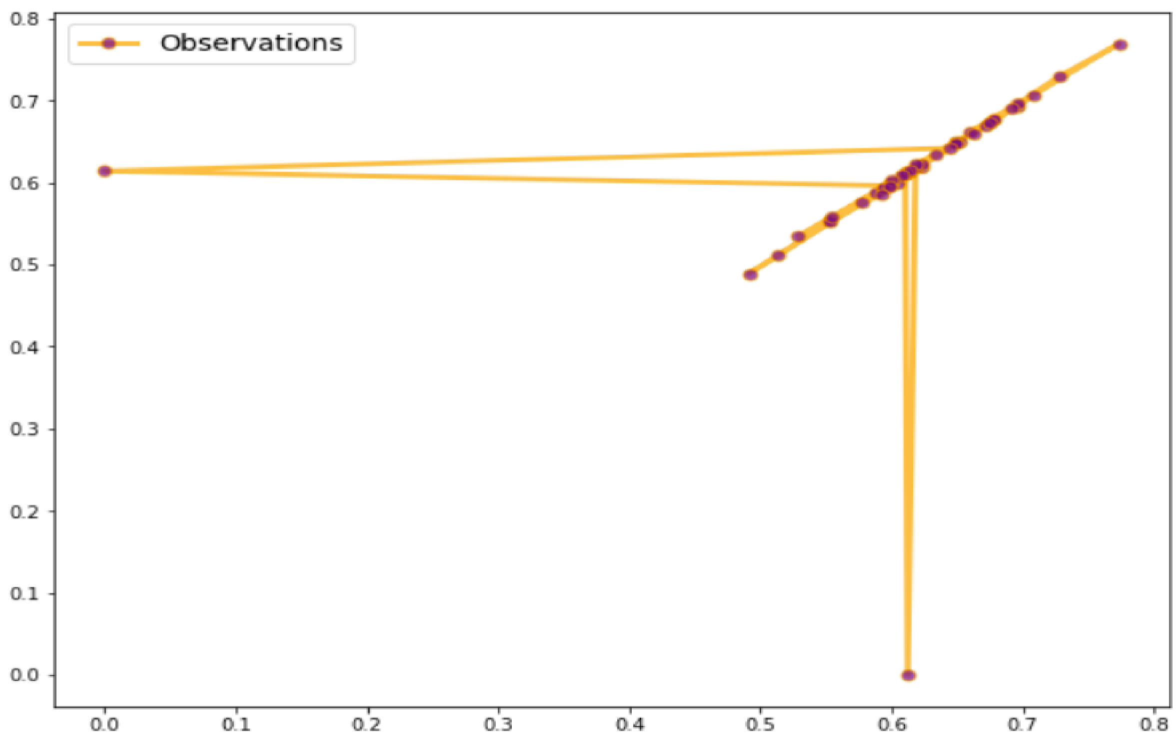


Testing (Generated):

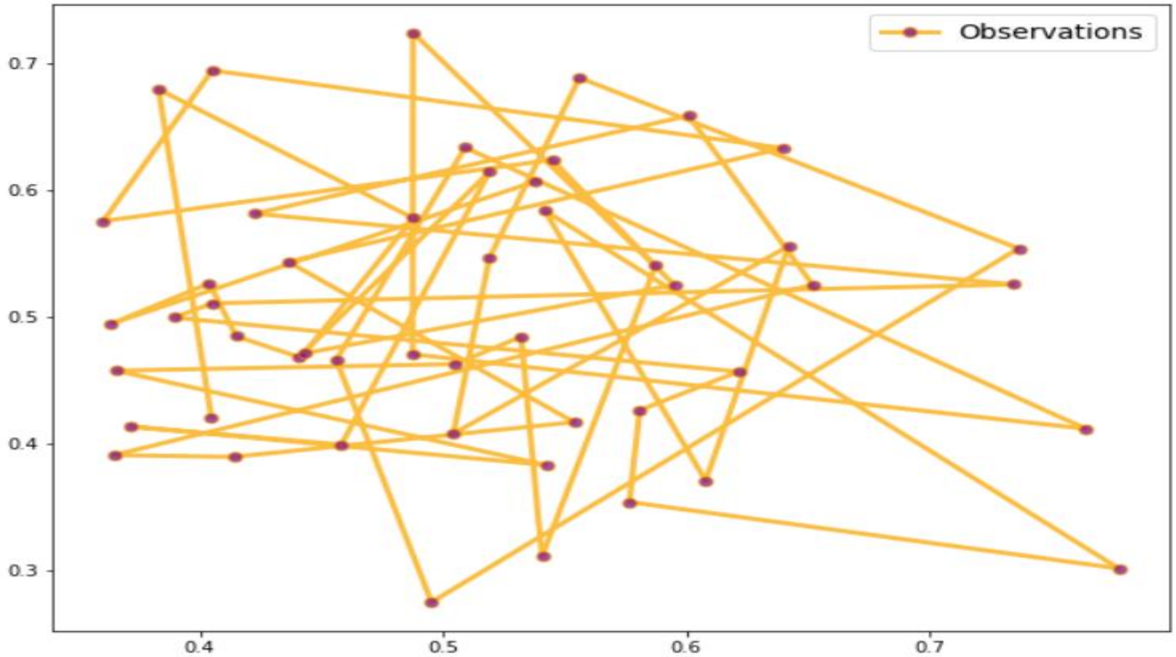


Model 7:

Training:



Testing (Generated):



The Forwards Algorithm:

```
[60] class HMM():
       def __init__(self, pi, A, B):
           self.pi_ = pi
           self.A_ = A
           self.B_ = B
           self.n_states_ = A.shape[0]

       def forwards(self, O):

         seq_length = O.shape[0]
         forward = np.zeros((self.n_states_, seq_length))

         # Initialization
         for s in range(self.n_states_):
           forward[s, 0] = self.pi_[s] * self.B_[s, O[0]]

         # Recursive step
         for t in range(1, seq_length):
           for s in range(self.n_states_):
             for sp in range(self.n_states_):
               forward[s, t] += forward[sp, t-1] * self.A_[sp, s] * self.B_[s, O[t]]

         #Termination
         forward_prob = np.sum(forward[:,-1])

         print(forward)

         return forward_prob
```