Rutuja Dukhande
1001730748
CSE 6363: Machine Learning

Assignment 2

Hidden Markov Models

- **Preparing the Data:**

The final dataset size, before splitting into training and test, is (350, 30, 4).

```
#left-hand centroid data
lc = df.iloc[:,:60].to_numpy()
all_lc_data = lc.reshape(350,30,2)


#right-hand centroid data
rc =df.iloc[:,390:450].to_numpy()
all_rc_data = rc.reshape(350,30,2)

all_data = np.concatenate((all_lc_data,all_lc_data), axis =2)

all_data.shape
```

✓ 0.1s

```
(350, 30, 4)
```

Splitting the data:

Split the data into training and testing

```
Model1_train_data, Model1_test_data = train_test_split( get_data[:50], test_size=0.10,random_state=42)
Model2_train_data, Model2_test_data = train_test_split( get_data[50:100], test_size=0.10, random_state=42)
Model3_train_data, Model3_test_data = train_test_split( get_data[100:150], test_size=0.10, random_state=42)
Model4_train_data, Model4_test_data = train_test_split( get_data[150:200], test_size=0.10, random_state=42)
Model5_train_data, Model5_test_data = train_test_split( get_data[200:250], test_size=0.10, random_state=42)
Model6_train_data, Model6_test_data = train_test_split( get_data[250:300], test_size=0.10, random_state=42)
Model7_train_data, Model7_test_data = train_test_split( get_data[300:350], test_size=0.10, random_state=42)
Model1_train_data.shape, Model1_test_data.shape
```

✓ 0.7s

```
((45, 30, 4), (5, 30, 4))
```

7 classes:

```
'ZoomIn':0,'ZoomOut':1, 'MoveLeft':2,'MoveRight': 3, 'MoveDown': 4, 'MoveUp':
5,'Press': 6
```

- **Normalizing the Data**

Normalization

```
def NormalizeData(data):
    return (data - np.min(data)) / (np.max(data) - np.min(data))
```

✓ 0.9s

```
get_data = NormalizeData(all_data)
get_data.shape
```

✓ 0.7s

```
(350, 30, 4)
```

```
get_data = NormalizeData(all_data)
print(get_data.shape)
print("Data after Normalization", get_data)
```

✓ 0.4s

```
Output exceeds the size limit. Open the full output data in a text editor
(350, 30, 4)
Data after Normalization [[[0.64631579 0.81894737 0.64631579 0.81894737]
  [0.54526316 0.54526316 0.54526316 0.54526316]
  [0.54526316 0.54526316 0.54526316 0.54526316]
  ...
  [0.39157895 0.38947368 0.39157895 0.38947368]
  [0.38947368 0.38526316 0.38947368 0.38526316]
  [0.38526316 0.38526316 0.38526316 0.38526316]]

 [[0.71578947 0.71578947 0.71578947 0.71578947]
  [0.71578947 0.71578947 0.71578947 0.71578947]
  [0.71368421 0.71368421 0.71368421 0.71368421]
  ...
  [0.37473684 0.37263158 0.37473684 0.37263158]
  [0.37263158 0.37263158 0.37263158 0.37263158]
  [0.37263158 0.37052632 0.37263158 0.37052632]]
```

- **Evaluating the Test Data:**

Evaluating the test set on the 7 trained models by computing the likelihood of each input test sample against all 7 models.

->Using Score function to evaluate the likelihood of a model given some input sequence.

```
def test(data, models):
    scores = [None] * len(models.values())
    mod ={}
    mod_num=1

    for i, m in enumerate(models.values()):
        scores[i] = m.score(data)
        mod[mod_num] = m
        mod_num+=1

    scores = [round(num, 4) for num in scores]
    return scores, mod
```

-> Given an input sample, the predicted class label corresponds to the model with the highest likelihood.

```
Input No of Test Data: 1
Scores:  [60.2007, -131.3746, 45.4115, 66.1354, 70.4735, 70.0948, 62.3962]
Highest likelihood: 4
-------------------------------------------------------------

Input No of Test Data: 2
Scores:  [103.5316, -5.2209, 46.6119, 101.4113, 84.7387, 84.9092, 66.4618]
Highest likelihood: 0
-------------------------------------------------------------

Input No of Test Data: 3
Scores:  [64.2975, -173.0833, 53.5201, 65.1623, 71.6499, 95.2245, 87.1991]
Highest likelihood: 5
-------------------------------------------------------------

Input No of Test Data: 4
Scores:  [157.5684, 164.8435, 47.6072, 146.2961, 102.5206, 103.2416, 70.3865]
Highest likelihood: 1
-------------------------------------------------------------

Input No of Test Data: 5
Scores:  [158.258, 146.4038, 50.1258, 145.2359, 102.6862, 110.6397, 78.0522]
Highest likelihood: 0
-------------------------------------------------------------

Input No of Test Data: 6
Scores:  [151.6293, 199.3864, 39.3498, 145.6529, 100.7418, 76.8911, 45.0336]
Highest likelihood: 1
-------------------------------------------------------------
```

As you can see for each input test sample it is calculating likelihood against all models from model1 to model 7 respectively.

For instance, for test data input sample 1, Index 4 is the highest likelihood among all other models so ‘model 4’ -> it corresponds to ‘MoveRight’ class label.

- **Finding the Best Configuration**

Conduct a hyperparameter search by training the models using a different number of components for each model. In your report, include a table that shows the test accuracy for each configuration you tried. Highlight the best performing model by showing its results in the table in bold.

Configuration 1

➔ **Training the models for Configuration1 using different no. of components**

```
#1. hyperparameter search by training the models using a different number of components for each model.

model1_com1 = GaussianHMM(n_components=2)
model1_com1.fit(Model1_train_data.reshape(-1,4))

model2_com1 = GaussianHMM(n_components=4)
model2_com1.fit(Model2_train_data.reshape(-1,4))

model3_com1 = GaussianHMM(n_components=9)
model3_com1.fit(Model3_train_data.reshape(-1,4))

model4_com1 = GaussianHMM(n_components=12)
model4_com1.fit(Model4_train_data.reshape(-1,4))

model5_com1 = GaussianHMM(n_components=11)
model5_com1.fit(Model5_train_data.reshape(-1,4))

model6_com1 = GaussianHMM(n_components=8)
model6_com1.fit(Model6_train_data.reshape(-1,4))

model7_com1 = GaussianHMM(n_components=14)
model7_com1.fit(Model7_train_data.reshape(-1,4))
```

```
accuracy = accuracy(pred_for_Configuration_1)
print("Test Accuracy for configuration1", accuracy, "%")
```

```
✓ 0.5s
```

```
Test Accuracy for configuration1 60.0 %
```

Configuration 2

➔ **Training the models for Configuration2 using different no. of components**

```
#2.Configuration2: hyperparameter search by training the models using a different number of components for each model.

model1_com2 = GaussianHMM(n_components=11)
model1_com2.fit(Model1_train_data.reshape(-1,4))

model2_com2 = GaussianHMM(n_components=2)
model2_com2.fit(Model2_train_data.reshape(-1,4))

model3_com2 = GaussianHMM(n_components=14)
model3_com2.fit(Model3_train_data.reshape(-1,4))

model4_com2 = GaussianHMM(n_components=1)
model4_com2.fit(Model4_train_data.reshape(-1,4))

model5_com2 = GaussianHMM(n_components=6)
model5_com2.fit(Model5_train_data.reshape(-1,4))

model6_com2 = GaussianHMM(n_components=7)
model6_com2.fit(Model6_train_data.reshape(-1,4))

model7_com2 = GaussianHMM(n_components=21)
model7_com2.fit(Model7_train_data.reshape(-1,4))
```

```
#accuracy for config2
accuracy2 = accuracy(pred_for_Configuration_2)
print("Test Accuracy for configuration2", accuracy2, "%")
```

✓ 0.7s

```
Test Accuracy for configuration1 40.0 %
```

Configuration 3

➔ **Training the models for Configuration3 using different no. of components**

```
#3. Configuration 3: hyperparameter search by training the models using a different number of components for each model.

model1_com3 = GaussianHMM(n_components=1)
model1_com3.fit(Model1_train_data.reshape(-1,4))

model2_com3 = GaussianHMM(n_components=21)
model2_com3.fit(Model2_train_data.reshape(-1,4))

model3_com3 = GaussianHMM(n_components=9)
model3_com3.fit(Model3_train_data.reshape(-1,4))

model4_com3 = GaussianHMM(n_components=13)
model4_com3.fit(Model4_train_data.reshape(-1,4))

model5_com3 = GaussianHMM(n_components=22)
model5_com3.fit(Model5_train_data.reshape(-1,4))

model6_com3 = GaussianHMM(n_components=4)
model6_com3.fit(Model6_train_data.reshape(-1,4))

model7_com3 = GaussianHMM(n_components=8)
model7_com3.fit(Model7_train_data.reshape(-1,4))
```

```
#accuracy for config3
accuracy3 = accuracy(pred_for_Configuration_3)
print("Test Accuracy for configuration3", accuracy3,"%")
```

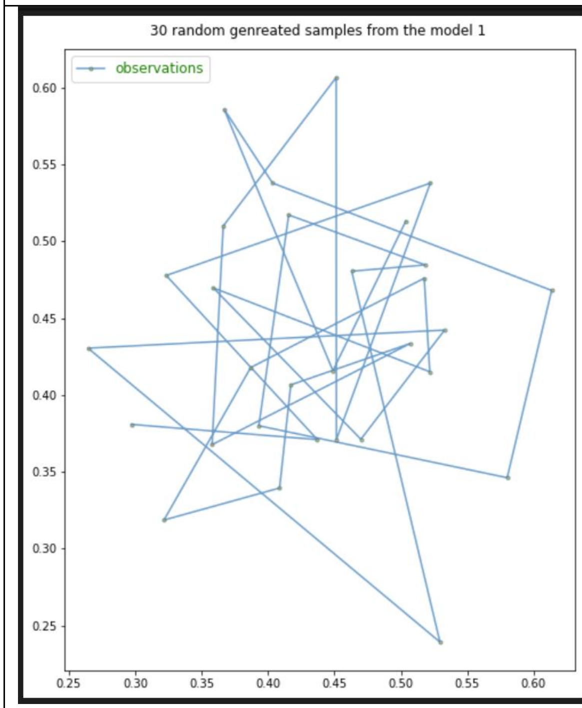
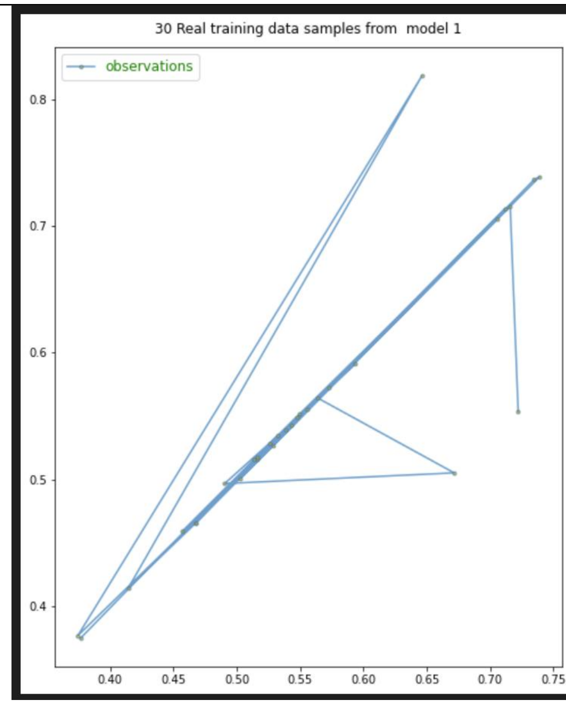
[195] ✓ 0.5s

```
Test Accuracy for configuration3 51.42857142857142 %
```

Testing Accuracy:

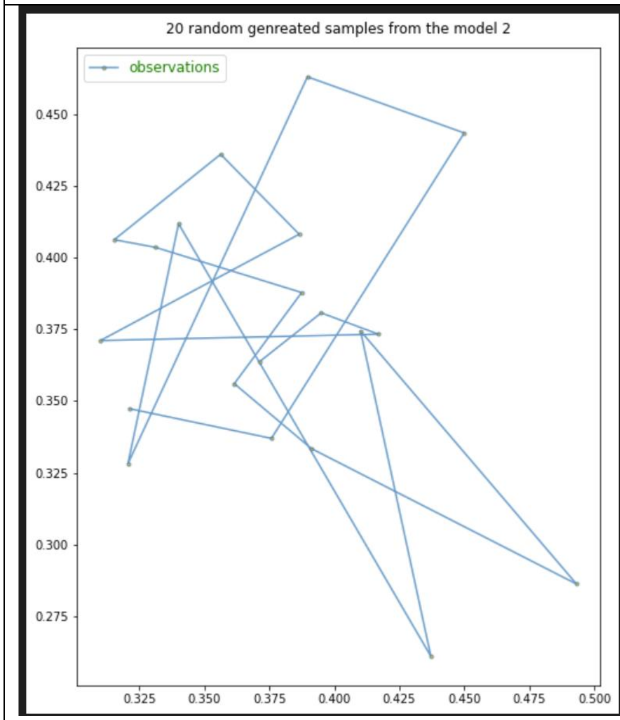
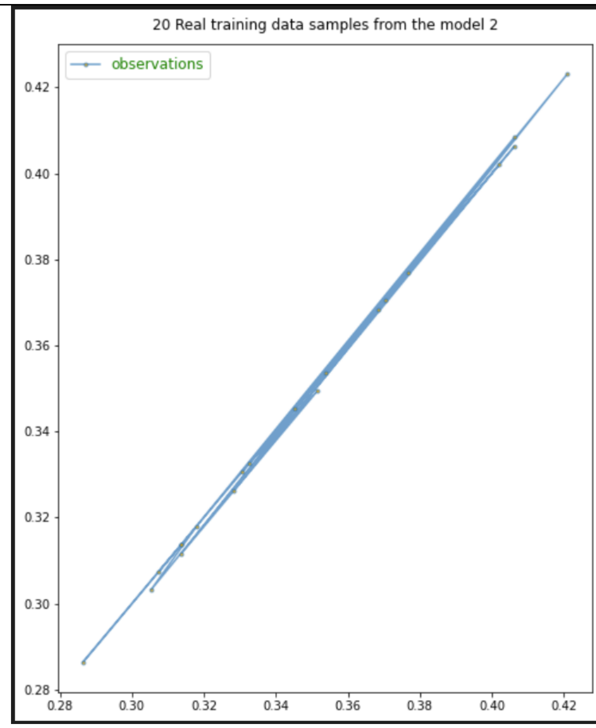
Configuration	**Testing Accuracy**
Configuration 1	**60 %**
Configuration 2	**40%**
Configuration 3	**51.42%**

- **Sampling from the HMM**
    1. **Model 1: ZoomIn**

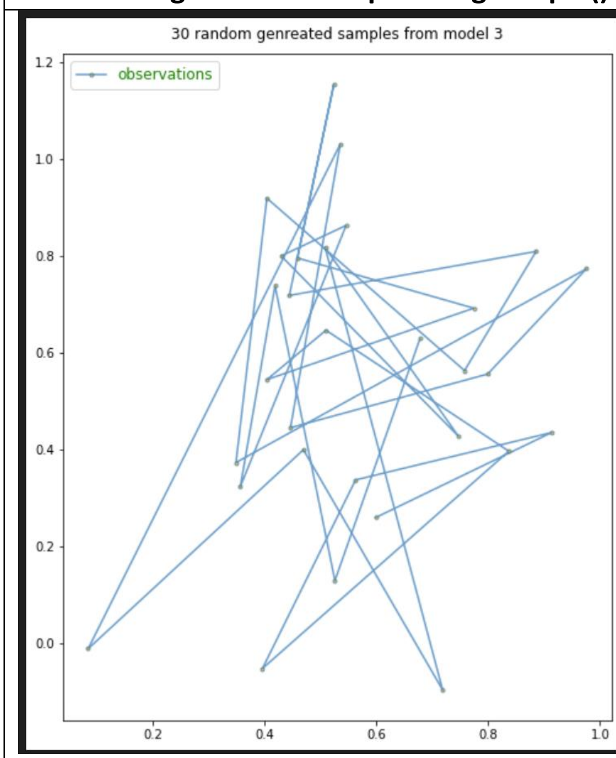
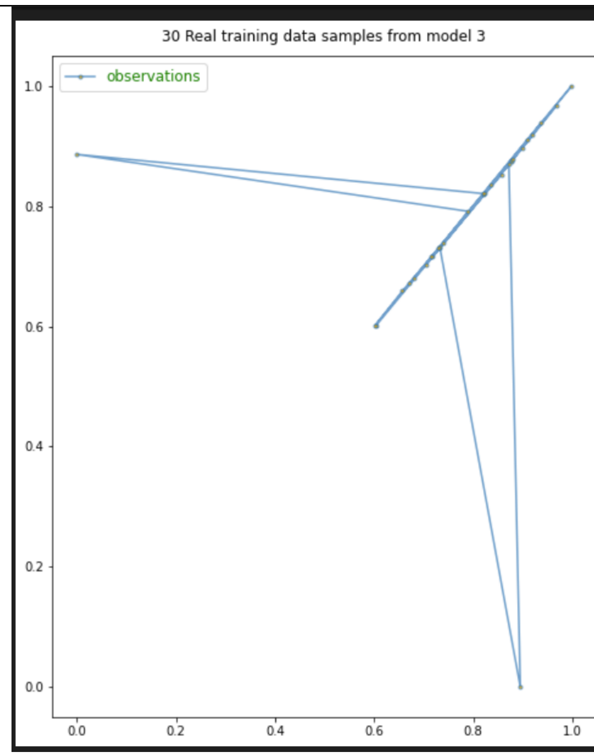
<table>
<tr><td>30 random generated sample using sample()</td><td>30 training samples from model 1</td></tr>
<tr><td>
</td><td>
</td></tr>
</table>

- ➢ The plot shows the sequence of samples generated with the transitions between them.
- ➢ First column represents random sample generated from the models with sample(). And second column represents the transition from one state to another states of training data.
- ➢ We can see that, there are no transition between some of the samples in randomly generated samples than when generated plot for training data. This is consistent for all other models.

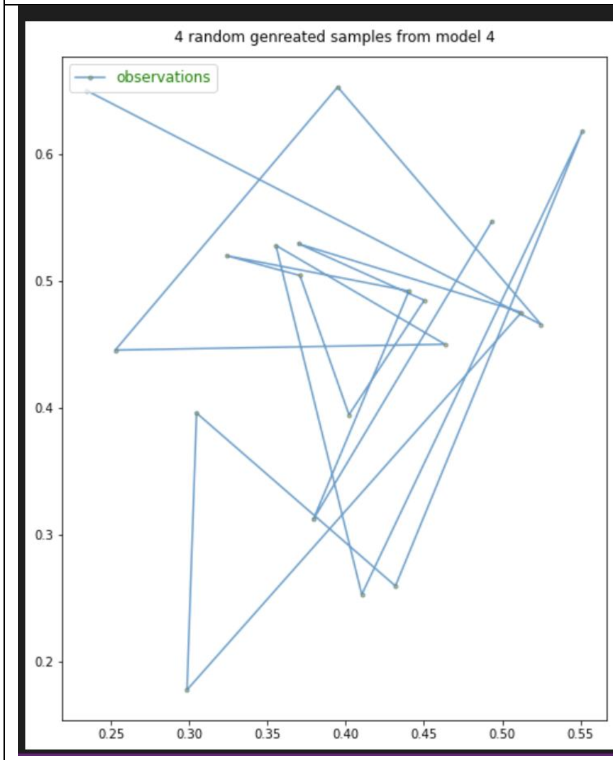
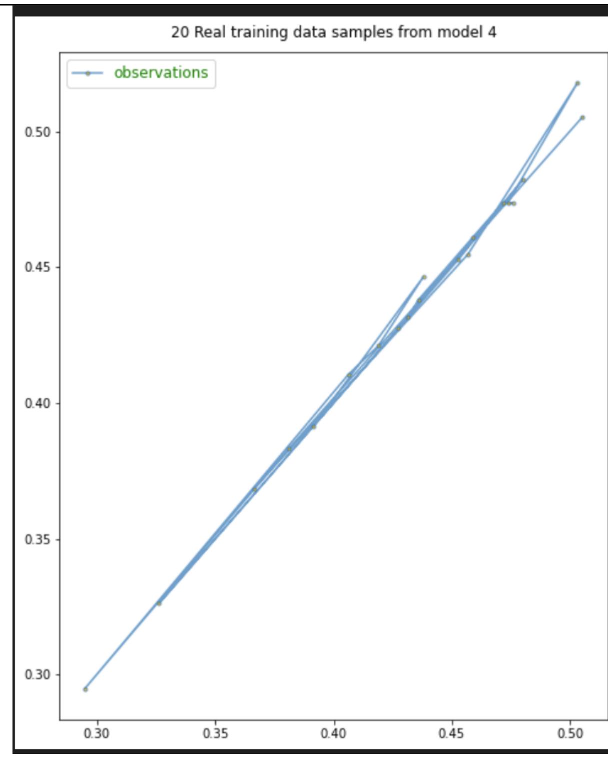
2. Model 2: ZoomOut

4 random generated sample using sample()	4 training samples from model 2
	

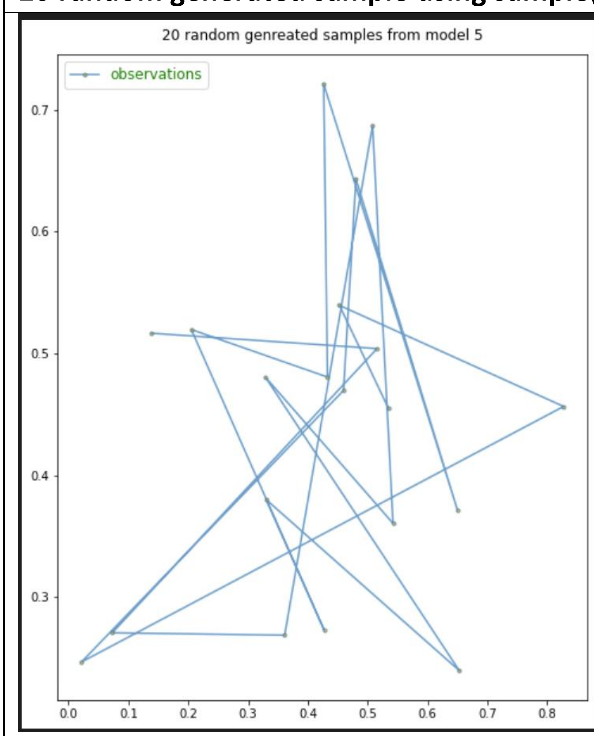
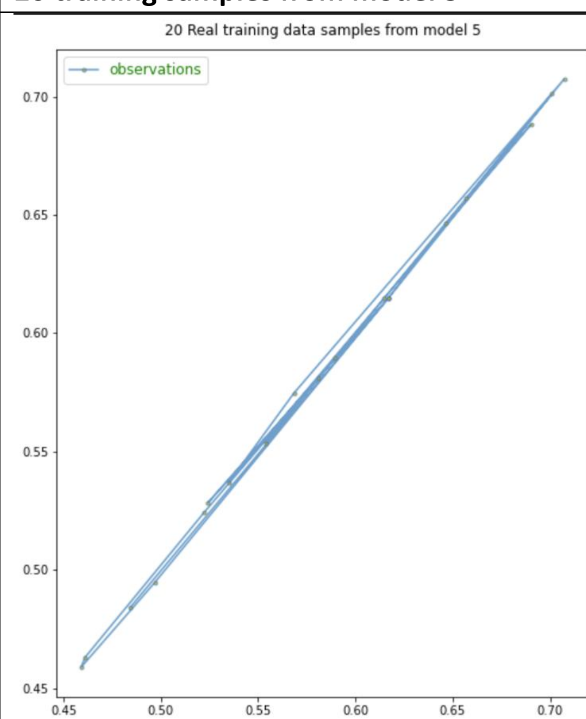
3. Model 3: MoveLeft

30 random generated sample using sample()	30 training samples from model 3
	

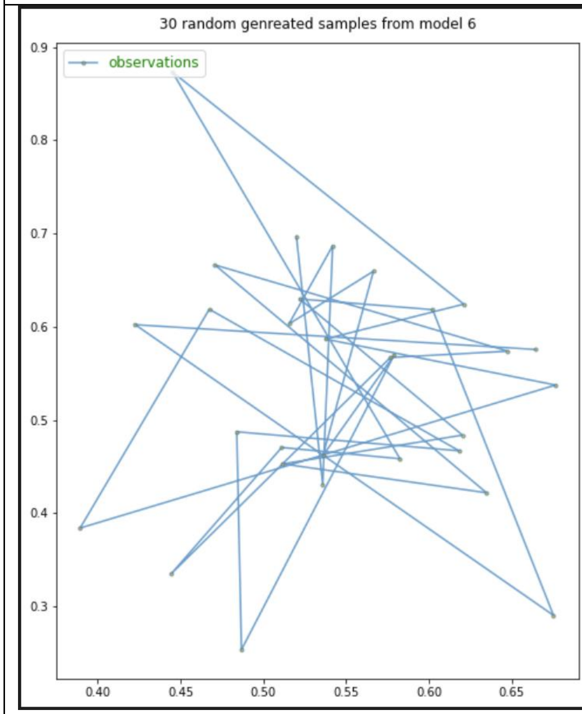
4. **Model 4: MoveRight**

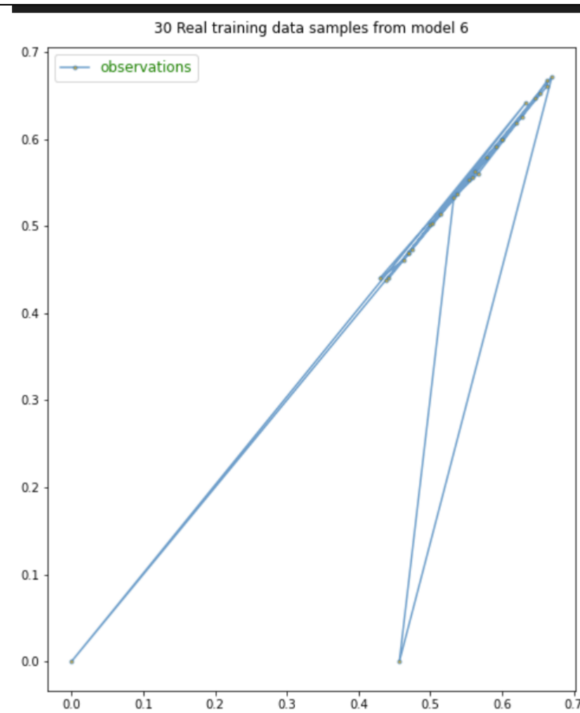
<table>
<tr><th>20 random generated sample using sample()</th><th>20 training samples from model 4</th></tr>
<tr><td>
</td><td>
</td></tr>
</table>

5. **Model 5: MoveDown**

<table>
<tr><th>20 random generated sample using sample()</th><th>20 training samples from model 5</th></tr>
<tr><td>
</td><td>
</td></tr>
</table>

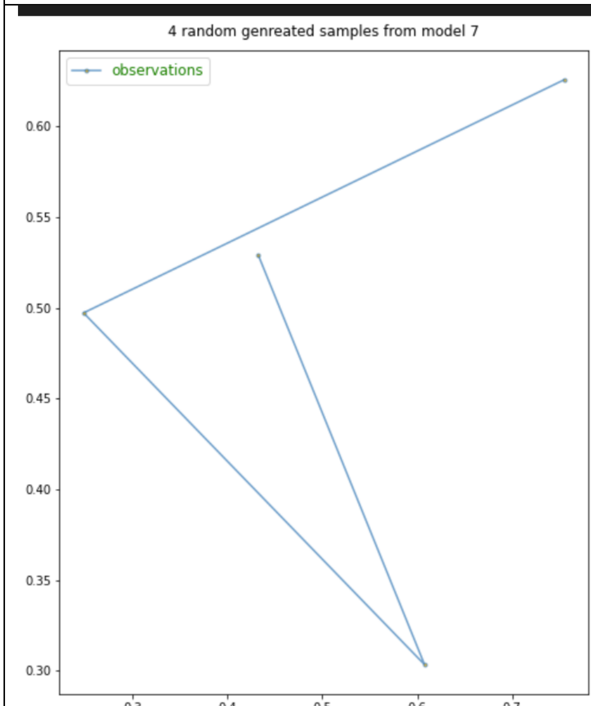
6. Model 6: MoveUp

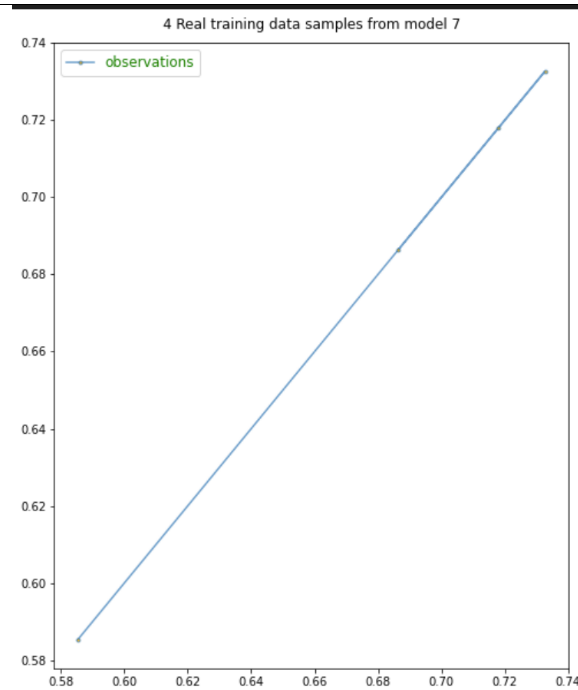
<table>
<tr><th>30 random generated sample using sample()</th><th>30 training samples from model 6</th></tr>
<tr><td>


</td><td>


</td></tr>
</table>

7. Model 7: Press

<table>
<tr><th>4 random generated sample using sample()</th><th>4 training samples from model 7</th></tr>
<tr><td>


</td><td>


</td></tr>
</table>