JUNAID GIRKAR

60004190057

TE COMPS A4

**EXPERIMENT - 6**

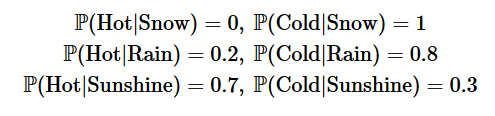
**AIM**: Implement Hidden Markov Model.

**THEORY**:

A Hidden Markov Model (HMM) is a statistical model which is also used in machine learning. It can be used to describe the evolution of observable events that depend on internal factors, which are not directly observable. These are a class of probabilistic graphical models that allow us to predict a sequence of unknown variables from a set of observed variables.

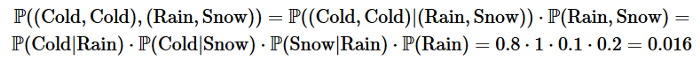
In a Hidden Markov Model (HMM), we have an invisible Markov chain (which we cannot observe), and each state generates in random one out of k observations, which are visible to us.

Let’s look at an example. Suppose we have the Markov Chain from above, with three states (snow, rain and sunshine), P - the transition probability matrix and q — the initial probabilities. This is the invisible Markov Chain — suppose we are home and cannot see the weather. We can, however, feel the temperature inside our room, and suppose there are two possible observations: hot and cold, where:



**Basic Example**

As a first example, we apply the HMM to calculate the probability that we feel cold for two consecutive days. In these two days, there are 3\*3=9 options for the underlying Markov states. Let us give an example for the probability computation of one of these 9 options:



**Finding Hidden States**

In some cases we are given a series of observations, and want to find the most probable corresponding hidden states.

A brute force solution would take exponential time (like the calculations above); A more efficient approach is called the Viterbi Algorithm; its main idea is as follows: we are given a sequence of observations o₁,…,oₜ . For each state i and t=1,…,T, we define



That is, the maximum probability of a path which ends at time t at the state i, given our observations. The main observation here is that by the Markov property, if the most likely path that ends with i at time t equals to some i\* at time t−1, then i\* is the value of the last state of the most likely path which ends at time t−1. This gives us the following forward recursion:



here, αⱼ(oₜ) denotes the probability to have oₜ when the hidden Markov state is j

**Application of Hidden Markov Model**

An application, where HMM is used, aims to recover the data sequence where the next sequence of the data can not be observed immediately but the next data depends on the old sequences. Taking the above intuition into account the HMM can be used in the following applications:

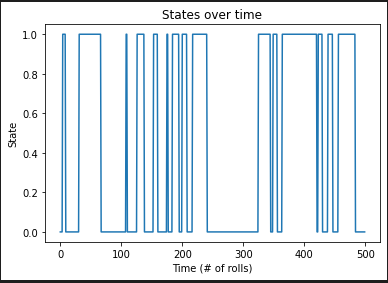
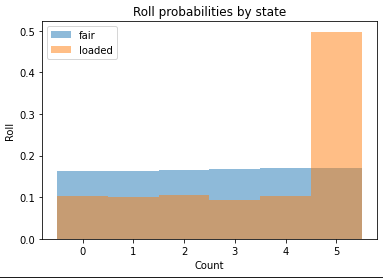
* Computational finance
* speed analysis
* Speech recognition
* Speech synthesis
* Part-of-speech tagging
* Document separation in scanning solutions
* Machine translation
* Handwriting recognition
* Time series analysis
* Activity recognition
* Sequence classification
* Transportation forecasting

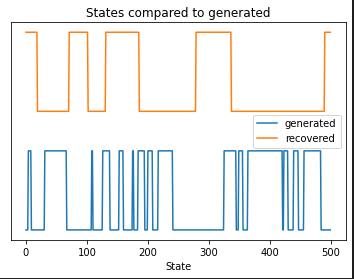
CODE:

| import numpy as np import matplotlib.pyplot as plt  from hmmlearn import hmm  # Now, let's act as the casin and exchange a fair die for a loaded one # and generate a series of rolls that someone at the casino would # observe.  # make our generative model with two components, a fair die and a # loaded die gen\_model = hmm.MultinomialHMM(n\_components=2, random\_state=99)  # the first state is the fair die so let's start there so no one # catches on right away gen\_model.startprob\_ = np.array([1.0, 0.0])  # now let's say that we sneak the loaded die in: # here, we have a 95% chance to continue using the fair die and a 5% # chance to switch to the loaded die # when we enter the loaded die state, we have a 90% chance of staying # in that state and a 10% chance of leaving gen\_model.transmat\_ = np.array([[0.95, 0.05],  [0.1, 0.9]])  # now let's set the emission means: # the first state is a fair die with equal probabilities and the # second is loaded by being biased toward rolling a six gen\_model.emissionprob\_ = \  np.array([[1 / 6, 1 / 6, 1 / 6, 1 / 6, 1 / 6, 1 / 6],  [1 / 10, 1 / 10, 1 / 10, 1 / 10, 1 / 10, 1 / 2]])  # simulate the loaded dice rolls rolls, gen\_states = gen\_model.sample(30000)  # plot states over time, let's just look at the first rolls for clarity fig, ax = plt.subplots() ax.plot(gen\_states[:500]) ax.set\_title('States over time') ax.set\_xlabel('Time (# of rolls)') ax.set\_ylabel('State') fig.show()  # plot rolls for the fair and loaded states fig, ax = plt.subplots() ax.hist(rolls[gen\_states == 0], label='fair', alpha=0.5,  bins=np.arange(7) - 0.5, density=True) ax.hist(rolls[gen\_states == 1], label='loaded', alpha=0.5,  bins=np.arange(7) - 0.5, density=True) ax.set\_title('Roll probabilities by state') ax.set\_xlabel('Count') ax.set\_ylabel('Roll') ax.legend() fig.show()  # split our data into training and validation sets (50/50 split) X\_train = rolls[:rolls.shape[0] // 2] X\_validate = rolls[rolls.shape[0] // 2:]  # check optimal score gen\_score = gen\_model.score(X\_validate)  best\_score = best\_model = None n\_fits = 50 np.random.seed(13) for idx in range(n\_fits):  model = hmm.MultinomialHMM(  n\_components=2, random\_state=idx,  init\_params='se')   model.transmat\_ = np.array([np.random.dirichlet([0.9, 0.1]),  np.random.dirichlet([0.1, 0.9])])  model.fit(X\_train)  score = model.score(X\_validate)  print(f'Model #{idx}\tScore: {score}')  if best\_score is None or score > best\_score:  best\_model = model  best\_score = score  print(f'Generated score: {gen\_score}\nBest score: {best\_score}')  # use the Viterbi algorithm to predict the most likely sequence of states # given the model states = best\_model.predict(rolls)  # plot our recovered states compared to generated (aim 1) fig, ax = plt.subplots() ax.plot(gen\_states[:500], label='generated') ax.plot(states[:500] + 1.5, label='recovered') ax.set\_yticks([]) ax.set\_title('States compared to generated') ax.set\_xlabel('Time (# rolls)') ax.set\_xlabel('State') ax.legend() fig.show()   print(f'Transmission Matrix Generated:\n{gen\_model.transmat\_.round(3)}\n\n'  f'Transmission Matrix Recovered:\n{best\_model.transmat\_.round(3)}\n\n')   print(f'Emission Matrix Generated:\n{gen\_model.emissionprob\_.round(3)}\n\n'  f'Emission Matrix Recovered:\n{best\_model.emissionprob\_.round(3)}\n\n') |
| --- |

OUTPUT:

| Model #0 Score: -26391.3688134072 Model #1 Score: -26395.55036572371 Model #2 Score: -26405.788668012425 Model #3 Score: -26396.290282611586 Model #4 Score: -26395.550365729432 Model #5 Score: -26375.803386308733 Model #6 Score: -26395.484479933773 Model #7 Score: -26300.67439789695 Model #8 Score: -26265.231805566247 Model #9 Score: -26395.550355030126 Model #10 Score: -26317.466352042524 Model #11 Score: -26405.60102609709 Model #12 Score: -26254.60496774158 Model #13 Score: -26395.48205064964 Model #14 Score: -26247.84864999629 Model #15 Score: -26279.11283863862 Model #16 Score: -26236.968567769647 Model #17 Score: -26320.826969997557 Model #18 Score: -26273.891661064114 Model #19 Score: -26405.955888278735 Model #20 Score: -26405.478747288675 Model #21 Score: -26385.9158985128 Model #22 Score: -26395.485391535083 Model #23 Score: -26395.550366098352 Model #24 Score: -26308.405549046784 Model #25 Score: -26395.550231732344 Model #26 Score: -26296.282172066236 Model #27 Score: -26382.50004338152 Model #28 Score: -26394.036977133965 Model #29 Score: -26396.28994962292 Model #30 Score: -26297.051796993994 Model #31 Score: -26282.17385580235 Model #32 Score: -26315.982257345328 Model #33 Score: -26255.807459691157 Model #34 Score: -26309.4618644052 Model #35 Score: -26395.550365831146 Model #36 Score: -26253.05848049172 Model #37 Score: -26395.550513487622 Model #38 Score: -26395.48290593895 Model #39 Score: -26276.364463630252 Model #40 Score: -26395.55036572371 Model #41 Score: -26395.48180629254 Model #42 Score: -26257.78147660928 Model #43 Score: -26395.550374348983 Model #44 Score: -26282.339321529027 Model #45 Score: -26262.651049578308 Model #46 Score: -26344.068596316894 Model #47 Score: -26229.219361702373 Model #48 Score: -26392.68971014984 Model #49 Score: -26337.637199573255 Generated score: -26230.575868403906 Best score: -26229.219361702373  Transmission Matrix Generated: [[0.95 0.05]  [0.1 0.9 ]]  Transmission Matrix Recovered: [[0.922 0.078]  [0.059 0.941]]   Emission Matrix Generated: [[0.167 0.167 0.167 0.167 0.167 0.167]  [0.1 0.1 0.1 0.1 0.1 0.5 ]]  Emission Matrix Recovered: [[0.106 0.108 0.114 0.105 0.113 0.454]  [0.168 0.168 0.167 0.171 0.172 0.153]] |
| --- |



**CONCLUSION:** We learnt about the Hidden Markov Model and implemented it in python using the library HmmLearn.