Lisa Meng
Shagun Gupta
INF 522
Homework 7

REPORT

Part 1: Implementation

The Viterbi algorithm was implemented to decode the noisy distances to find the robot's trajectory.

The Data Structures used to implement the algorithm were mostly lists and dictionaries. The numpy and math libraries were used extensively to handle data dimensions, multiplication and other operations. The matrices created to handle the HMM code were:

```
min_noise = 0.7
max_noise = 1.3
t1 #the initial probability table for each state
states, valid_states #Each position(x,y) was treated as a state and valid states were used to store if given state was 1(valid) or 0(invalid)
tran_matrix #The 100x100 matrix gives the prob of going from state 1 to state 2. It is 0 for invalid states.
possible_observations, euclidean_dist_matrix #All possible values that can be observed in the evidence matrix between [0.7d, 1.3d] and the distance matrix based on euclidean distance for all states and towers.
Seq #The final sequence of states that the robot moved through
```

To implement the code optimally, functions were used to avoid any redundant code. The following functions were used:

```
get_valid_states()#returns states, valid_states
get_intitial_prob()#returns t1
get_prob()# returns prob of going from position 1 to position 2 on the grid
get_cpt()#returns tran_matrix
get_distance()#returns possible_observations, euclidean_dist_matrix
get_evidence_matrix()#returns evidence matrix for a time step given all noisy observations
HMM()#handles main implementation of viterbi algorithm, returns seq
```

The numpy library was used to optimize basic operations like finding max probability in a time step or multiplication amongst the various matrices. Data structures were used to store reusable variables instead of making function calls again and again in each step.

The HMM Viterbi algorithm is quite confusing and while we studied the algorithm for boolean variables it was a little difficult to visualise the data structures that were needed to implement the algorithm. Many times 3-dimensional matrices were encountered and it was a little challenging to handle multi-dimensional data. However the assignment became more and more clear as we progressed through the different steps of the algorithm.

The final most probable trajectory followed by the robot:
Time 1 : (5, 3)
Time 2 : (6, 3)
Time 3 : (7, 3)
Time 4 : (7, 2)
Time 5 : (7, 1)
Time 6 : (7, 2)
Time 7 : (7, 1)
Time 8 : (6, 1)
Time 9 : (5, 1)
Time 10 : (4, 1)
Time 11 : (3, 1)

Part 2: Software Familiarization

After going through many libraries and research papers it was observed that multivariate observations are a special modification of the standard HMM problem that uses 1 observation variable in a sequence to decode the most probable state sequence and so no library exists that handles this modified problem directly.
Libraries handle the following sequence of observations:
['A','B','A']
And not:
[['A','A'],['A','B'],['B','A']]
Which is the type of problem we dealt with in Part 1.

However, to understand the functioning of the standard HMM model, the Hidden_Markov.hmm API was studied.

The API takes in the following parameters:

1. states (A list or tuple) – The set of hidden states : **states in Part 1**
2. observations (A list or tuple) – The set unique of possible observations: **possible_observations in Part 1**
3. start_prob (Numpy matrix, dimension = [length(states) X 1]) – The start probabilities of various states, given in the same order as 'states' variable. start_prob[i] = probability(start at states[i]): **t1 in Part1**

4. trans_prob (Numpy matrix, dimension = [len(states) X len(states)]) – The transition probabilities, with ordering same as ‘states’ variable . trans_prob[i,j] = probability(states[i] -> states[j]): **tran_matrix in Part 1**
5. em_prob (Numpy matrix, dimension = [len(states) X len(observations)]) – The emission probabilities, with ordering same as ‘states’ variable and ‘observations’ variable. em_prob[i,j] = probability(states[i],observations[j]): **euclidean_dist_matrix in Part 1**

For our problem statement in part 1 the dimension of euclidean_dist_matrix is : no. of observations (4) * no. of states (100) * no. of possible observations (166)

The library however accepts the dimension mentioned above.

The API class is called by passing the above parameters as:
Model = hmm(parameters)

Viterbi algorithm is implemented using a sequence of observations which in our case is a 2D array but the library is equipped to handle only 1D list/tuple

model.viterbi(obs[‘A’,’B’,’A’])

The above returns a sequence of the most probable states for given observations.

Due to the inability of a library to resolve the multivariate problem directly, the question in the assignment is not able to be solved using a library, thus it is difficult to determine how the performance of our code compares to that of a library.

Part 3: Applications

1. Computational finance: HMM models are applied to forecast regimes in market turbulence, inflation, and economic growth
2. Speech Recognition and synthesis: HMM is used to build a statistical model to infer the text sequences and use these labels for speech synthesis
3. Gene prediction: genomic sequences are used to identify genes, a method proven much more effective than using standard computational methods
4. Handwriting recognition: word, document and character images are used as sequences of observations for HMM models which produce 100% accuracy in most cases
5. Time series analysis: Time series problems are broken into various data points that are used as states while the data observed is used as evidence. This can be used to predict a time series sequence for as many states as needed.
6. Activity Recognition: A field that many struggle with in computer vision, HMMs resolve this complexity and help in monitoring, prediction and studying of human activity data

Contributions:

- For Part 1, Lisa and Shagun collectively discussed the concepts of Hidden Markov Models and the steps to implement the Viterbi algorithm for a given set of observations. We decided to initially code the algorithms individually. From the individual work, we then picked the best code to submit, which was Shagun's code.
- For Part 2, Lisa researched and implemented the Hidden_markov package for Viterbi algorithm.
- Part 3 was researched and completed by Shagun.
- The report was compiled by Lisa and Shagun.