HOMEWORK #7
HIDDEN MARKOV MODELS

Group Members: Onzali Suba (9690-3228-04)
and Nitya Pydipati (3219-5181-98)

Comparison with Scikit Learn-hmmlearn

- Machine Learning Library in Python

- HMMlearn follows scikit-learn API as close as possible, but adapted to sequence data.
- HMMlearn deals with various kinds of models where emission types include Gaussian, Gaussian Mixture and Multinomial whereas our program deals with the matrix given and just one type of emission.
- Our program uses the Viterbi algorithm whereas the HMMlearn allows us to use three types of algorithm, depending on the problem statement, namely – Viterbi, Forward-Backward algorithm and Baum-Welch algorithm.
- The HMMlearn package also allows you to work with multiple sequences whereas our algorithm works with just the one provided.
- HMMLearn is also built with Matplotlib thus it helps us visualize the emissions that are generated.
- HMMLearn also has a base class called **hmmlearn.base** which allows for easy evaluation of, sampling from, and maximum a posteriori estimation of the parameters of a HMM. It allows us to deal with various other kinds of emission data unlike the single type of data that our algorithm deals with

Applications of HMM

- DNA sequences can be considered as texts in the alphabet of four letters that represent the nucleotides. The difference in stochastic properties of Markov chain models for coding and non-coding regions of DNA can be used for gene finding. GeneMark is the first system that implemented this approach
- Analysis and design of queues and queuing networks: It covers a range of problems from productivity analysis of a carwash station to the design of optimal computer and telecommunication networks.
- Stochastic language modeling: It was shown that alphabet level Markov chains of order 5 and 6 can be used for recognition texts in different languages. The word level Markov chains of order 1 and 2 are used for language modeling for speech recognition
- HMM models are widely used in speech recognition, for translating a time series of spoken words into text.
- Model the correlations between the activities and the observed sensor data.

Structures used in the algorithms

- **Lists:**
  We use it in throughout the code for storing and retrieving the free cell values, noisy distances and the tower locations. Also, to determine prob states, determining and manipulating for probabilities.

```
distancetotower=[]
dist.append([euclidean_dist*0.7,euclidean_dist*1.3])
```

- **Dictionaries:**
  We use it in order to store free cell locations that are allowed in each timestep based on the condition (0.7d, 1.3d), timesteps corresponding to each valid free cell and the probability distributions for each free cell and its neighboring cells. Also, in many other instances of our algorithm.

```
dic = {}
dic[timestep][nei] = {}
dic[timestep][nei]['parent'] = items
present_prob = dic[timestep - 1][items]['prob'] * trans_prob[items][nei]
```

Challenges faced

- Reading the data
- Understanding how to calculate the probability distribution table in an efficient manner

Optimizations

- **Using enumerate:**
  It allows us to loop over something and have an automatic counter.

```
for j, tower in enumerate(tower_loc):
euclidean_dist = math.sqrt(pow(free[0]-tower[0],2) + pow(free[1]-tower[1],2))
dist.append([euclidean_dist*0.7,euclidean_dist*1.3])
```

- **Used xrange instead of range for better performance (since iteration through large no)**

```
for timestep in xrange(1,len(noisy_dist)):
```

Output

[(5, 5), (5, 4), (6, 4), (7, 4), (7, 3), (7, 2), (7, 1), (6, 1), (5, 1), (4, 1), (3, 1)]

Contributions

Code: Colloborated

Report: Colloborated