Homework 5 - Advanced NLP

Author: Prateek Srivastava

Problem 1:

1. The HMM table would look something like the following with the blank boxes having some properties:

	t=0	t=1	t=2	t=3	t=4	t=5
	u	c	p	c	p	c
a	P(a) * P(a | u) * P(u | start)	...				
b	P(b) * P(b | u) * P(u | start)	...				
c						
:						
:						

These encrypted alphabets in the word "ucpcpc" will be emitting "banana" respectively based on the probabilistic model which we will have in the corpus in the normal english language. The number of states in the HMM would be the number of alphabets i.e. 26.

1. Emission probabilities would be distributed like a gaussian distribution for a well trained HMM.

2. Assuming the encripted data would be small, it makes more sense to leave the model to a bigram model instead of a trigram or higher n-gram model. It will keep the model dense enough to make probabilistic predictions.

    - Advantages of trigram or higher model:
        - Probable better results for Maximum Likelihood Estimation i.e. hopefully less error rate
        - More data i.e. better numerical sparsity in terms of entropy
    - Disadvantages:
        - If data is less, won't be able to get substantially better results as compared to bigram model.

- More computation intensive as the model goes to a higher n-gram model.

Problem 2:

```
import numpy as np
from pprint import pprint

text = "a myth is a female moth"
initp = np.array([.45, .35, .15, .05])
tp = np.array([[.03, .42, .5, .05], [.01, .25, .65, .09], [.07, .03, .15, .75], [.3
ep = np.array([[.84, .05, .03, .05], [.01, .1, .45, .1], [.02, .02, .02, .6], [.84,

# forward algo
forward = np.zeros((6, 4))
for t in range(0, len(text.split())):
    if t == 0:
        tmp = np.dot(initp, tp)
        forward[t] = tmp * ep[t]
    else:
        tmp = np.dot(forward[t - 1], tp)
        forward[t] = tmp * ep[t]
# pprint(forward)

# backward algo
backward = np.zeros((6, 4))
inp = np.array([1, 1, 1, 1])
l = len(text.split()) - 1
for t in range(l, -1, -1):
    if t == l:
        tmp = np.dot(inp, tp)
        backward[t] = tmp * ep[t]
    else:
        tmp = np.dot(backward[t + 1], tp)
        backward[t] = tmp * ep[t]
# pprint(backward)

# gamma matrix after smoothing alpha and beta values
gamma = np.zeros((6, 4))
gamma = forward * backward

print("alpha4(NN) is ", forward[3][2]) # alpha4(NN)
print("alpha3(VB) is ",forward[2][3]) # alpha3(VB)
print("alpha1(DT) is ",forward[0][0]) # alpha1(NN)
print("beta4(NN) is ",backward[3][2]) # beta4(NN)
print("beta2(NN) is ",backward[1][2]) # beta2(NN)
```

Output :

```
alpha4(NN) is  3.1383023898e-05
alpha3(VB) is  0.006678735225
alpha1(DT) is  0.0357
beta4(NN) is  0.001978659
beta2(NN) is  0.000150340169033
```