CSE 476/598 Intro to Natural Language Processing

Assignment 2

Rohan Jain

rjjain1@asu.edu

Solution 1

(a)

Refer to the code for the exhaustive() function

Input:I am running.

Output:[[‘PRP' 'VBP' 'VBG' '.']]

(b)

Input: You look around at professional ballplayers and nobody blinks an eye .

Output: [[‘PRP' 'VBP' 'RB' 'IN' 'JJ' 'NNS' 'CC' 'NN' 'VBZ' 'DT' 'NN' '.']]

Solution 2

(a)(Refer to the code file for commented code)

```
def decode(self, sent):
    maxstart1=-5000000000000
    maxcost=-50000000000
    logsum=0
    #self.userSentence=raw_input("Enter a Sentence").lower()
    self.userSentenceUp=sent
    self.userSentence=sent.lower()
    wordBreak=nltk.word_tokenize(self.userSentence)
    wordBreakUp=nltk.word_tokenize(self.userSentenceUp)
    x=np.zeros(shape=(2,len(wordBreak)), dtype=object)
  counter=0
```

```
j=0
for word in wordBreak:
    maxcost=-50000000000
    if counter==0:
        counter=counter+1
        for state in self.states:
            startCost=self.priors.logprob(state)+self.emissions[state].logprob(wordBreak[0])
            if startCost>maxstart1:
                maxstart1=startCost
                maxstartstate=state

        x[0][j]=maxstartstate
        x[1][j]=maxstart1
        j=j+1
    else:
        for state in self.states:
            cost=x[1][j-1]+self.transitions[x[0][j-1]].logprob(state)+self.emissions[state].logprob(word)
            if cost>maxcost:
                maxcost=cost
                maxstate=state

        x[0][j]=maxstate
        x[1][j]=maxcost
        j=j+1
for i in range(0,len(wordBreak)):
    #print wordBreak[i]#x[0][i]
    print "",wordBreakUp[i],"/",x[0][i],
print ""
```

Code Explanation:

The above given function decode() uses the Viterbi algorithm to find the most likely tag sequence for a give word.

It first stores the input sentence in the variable userSentence and also converts it into lower case characters. Next it tokenizes the word using ntlk.word_tokenize(userStentence). We then define the following array to store the best Viterbi path probability and the its associated tags for the Viterbi algorithm.

```
x=np.zeros(shape=(2,len(wordBreak)), dtype=object)
```

We then define the following loop that finds the initial best tag from the start state which gives the maximum Viterbi path probability value-

```
for word in wordBreak:
        maxcost=-50000000000
        if counter==0:
            counter=counter+1

            for state in self.states:
                startCost=self.priors.logprob(state)+self.emissions[state].logprob(wordBreak[0])
                if startCost>maxstart1:
                    maxstart1=startCost
                    maxstartstate=state

            x[0][j]=maxstartstate
            x[1][j]=maxstart1
            j=j+1
```

The above loop searches for the best Viterbi path for the first tag from the start state using self.priors.logprob(state).

At each iteration it checks whether maxstart1(stores the maximum probability value) is lesser than startCost(Viterbi path probability value for that particular state). If it is less then the max is changed to startCost.

This loops over all the states and finally we get the tag with the maximum Viterbi path probability from the start state. We then store the tag and the probability in an array.

The rest of the tags are calculated by the code below in a similar manner.

```
else:
            for state in self.states:
```

```
            cost=x[1][j-1]+self.transitions[x[0][j-1]].logprob(state)+self.emissions[state].logprob(word)
            if cost>maxcost:
                maxcost=cost
                maxstate=state


        x[0][j]=maxstate
        x[1][j]=maxcost
        j=j+1
```

Once all the tags and their probabilities are achieved we print them-

```
for i in range(0,len(wordBreak)):
        print wordBreak[i]
```

(b)

```
def tagViterbi(self):
      f=open('E:\ASU\Intro to NLP\HW\HW2\hw2_data\sentences.txt','rU');
   for self.line in f:
        self.decode(self.line)
```

The function tagViterbi() opens a file and reads every line. Each line is then passed to the decode function to get the maximum likely sequence for each line in the file.

The printing is the done in the following manner in the decode function to achieve an output of the form: This/DT is/VBZ a/DT sentence/NN.

```
for i in range(0,len(wordBreak)):
        #print wordBreak[i]#x[0][i]
        print "",wordBreakUp[i],"/",x[0][i],


      print ""
```

(c)

The most likely sequence for the sentence below found by the tagViterbi() method is-

Input- You look around at professional ballplayers and nobody blinks an eye .

Output-

```
Training HMM...
You/PRP look/VBP around/RB at/IN professional/JJ ballplayers/NNS and/CC nobody/NN blinks/V
BZ an/DT eye/NN ./.
```

The output is same of both the parts as we can also see below:

Output 1:

```
[[‘PRP' 'VBP' 'RB' 'IN' 'JJ' 'NNS' 'CC' 'NN' 'VBZ' 'DT' 'NN' '.' ]]
```

Output 2:

```
You/PRP look/VBP around/RB at/IN professional/JJ ballplayers/NNS and/CC nobody/NN blinks/V
BZ an/DT eye/NN ./.
```

(d)
Output:

```
The/JJ report/NN is/VBZ subject/JJ to/TO review/VB ./.
The/JJ balance/NN is/VBZ n't/RB being/VBG budgeted/RP for/IN the/DT coming/VBG year/NN ./.
We/PRP begin/VBP by/IN considering/VBG the/DT much/JJ simpler/NN case/NN of/IN the/DT M
arkov/JJ chain/NN ./.
Somewhere/NNS ,/, somebody/'' is/VBZ bound/-NONE- to/TO love/VB us/PRP ./.
None/NN of/IN the/DT Trujillo/JJ family/NN remains/VBZ ./.
```

As we can see there are few errors present in the above tag sequence. The errors are as follows-

n’t:RB
somebody: ''
bound:-NONE-

Words like is n’t is not specified an appropriate tag. Since the word is separated by a whitespace character. Therefore while tokenizing the tokenizer considers is n’t as two separate words as is and n’t. Therefore when we apply the Viterbi algorithm on a sentence containing these two words, two separate tags are assigned to is and n’t and therefore an error occurs.

Answer 3

It took me 3 days to complete this assignment