EXPLANATION A3 PART 1

Training - def train()

Function for implementing training and computing probabilities.

Variable description:

tags1: Contains the list of all the tags

tags_uniq: Contains all the unique tags

ps1: Initial Probability

This is probability of each tag occuring at first position. It is the frequency that each tag at first position by the total count. The values are stored in 'log' values since all calculations are performed in log to avoid underflow.

i.e. P(tag at first position)=Total occurrences of the tag at the first position/Total number of tags at first position.

ps1=log(P(tag at first position))

ps: State Probability

This is the probability of a tag occurring irrespective of the position except for the tags occuring at the first position. It is frequency of each tag divided by total number of tags. The values are stored in log

i.e. P(tag)=Total occurrences of the tag/Total number of tags

ps=log(P(tag))

pss: Transition probability P(Si+1|Si)

This is probability of two tags occurring in sequence. It is frequency of all tag pairs in sequence divided by total number of tags pairs. The values are stored in log. pss variable is a dictionary which contain pair of tags as pairs e.g.P[('tag1','tag2')]. Please note that 'pss' stores the probabilities that two tags occur in a sequence. This is P(S and Si+1). To find P(Si+1|Si) we have to subtract P(Si) from P(S and Si+1).

i.e. $P(tag_{i-1}, tag_i)$=Total occurrences of (tag_{i-1}, tag_i)/Sum of the total occurrences of all the pairs of tags

pss=log($P(tag_{i-1}, tag_i)$)

pws: **Denotes emission probability P(Wi|Si)**

The is the probability of each word given a tag. For all tags, frequency is calculated for associated words, the probability is calculated by diving the frequency of each word by total number of words associated with that tag. The values are stored in log. We are storing emission probabilities in a data frame “pws” where columns denote tags and rows denote words.

pc:

This is the probability that three tags occur in a sequence. The values are
stored in log.
This is P(S,Si+1,Si+2). To find P(Si+2|Si,Si+1) we have to subtract P(Si,Si+1)
from P(S,Si+1,Si+2).
This probability is calculated using the function ‘joint_probability’
pc= Total occurrences of (tagi-2,tagi-1,tagi)/Sum of the total occurrences of all the three tags appearing in succession.

pc=log (pc)

Posterior probability calculation - def posterior()

[**Simple]**
The posterior probability is calculated using Naive Bayes assumption, that each hidden variable is independent from every other hidden variable. Given a set of observed variables(words) and hidden variables (POS tags), for each word and corresponding tag, the posterior probability of each tag given a word is given by
P(tag|word) = (P(word|tag) * P(tag))/P(word)
We can ignore the denominator as we have to compare the score for each tag and consider the maximum.
i.e. P(tag|word) α (P(word|tag) * P(tag))
This is for one word and corresponding tag, for a given sequence, we have to multiply the posterior probabilities of all the word-tag combination
P(tag1|word1)*P(tag2|word2)…P(tagn|wordn) α (P(word1|tag1) * P(tag1)) * (P(word2|tag2) * P(tag2)) ….. (P(wordn|tagn) * P(tagn))
Also, since the probabilities are stored in logs, we convert the above equation
to the following by taking logs on both sides
Posterior probability of the given word-tag sequence = log(P(tag1|word1)) +
log(P(tag2|word2))…+log(P(tagn|wordn))
In the program the probability P(word|tag) is the emission probability(likelihood)
and is stored in the variable ‘pws’ and the the probability P(tag) is the probability
of the hidden variable(prior) and is stores in the variable ‘ps’

[HMM]
The probability of each element in a Hidden Markov Chain depend on the previous state(transition probability) and the probability of each word(emission probability) depends on the hidden state. The probability of a given sequence of words and hidden states(tags) is given by
Probaility = [P(tag1)*P(word1|tag1)]* [P(word2|tag2)*P(tag2|tag1)]….*[P(wordn|tagn)*P(tagn|tagn-1)]

The transition probabilities (e.g. P(tag2|tag1)) are stored in pss

[Complex]
According to figure 1c given in the question, the probability of state depends on the two previous states(complex probability) and the probability of each word(emission probability) depends on the hidden state. The probability of a given sequence of words is given by
Probability = [P(tag1)*P(word1|tag1)]*
[P(word2|tag2)*P(tag2|tag1)]*[P(word3|tag3)*P(tag3|tag2,tag1)]…* [P(wordn|tagn)*P(tagn|tagn-1,tagn-2)]
The complex probabilities (e.g. P(tag3|tag1,tag2)) are stored in 'pc'
Please note that 'pc' stores the probabilities that three tags occur in a sequence.

Algorithms

Naïve Bayes (simple) - def simplified()

In this method we have implemented part of speech tagging using simplified Bayes net method. This is implemented by maximising the posterior probability(check posterior probability section) for each tag given a word. In case the test data has an unseen word. A small probability is substituted for P(Word|Tag).

We calculate maximum tag score for corresponding word by iterating over word list and tag list. Tag score is given by:

tag_score=Hidden probability of the tag+Emission probability

= P(tag) + P(Wi|Si)

= ps+pws

We compare the tag score with previous computed value. The tag with the maximum score is labeled with the word.

Viterbi (HMM) - def hmm_viterbi()

VITERBI ALGORITHM:

We have implemented viterbi algorithm using dynamic programming and backtracking.

Variable description:

viterbi: Dataframe that will act as a viterbi table for storing probabilities.

backtrack: Dataframe that will act as backtracking table containing tags.

tags_uniq: List containing all the twelve unique tags.

words_dict: Dictionary of words with each word acting as a key.

pws: Probability of word|tag (word given tag)

ps1: Probability that the given tag is the first word of the sentence.

Step 1:

In the first for loop we first build the first column of the viterbi table by calculating the probabilities of each cell by changing the tag and keeping the word fixed i.e. viterbi[tag][words[0]]=pws+p1 or

for t = 1: vj (1) = wj ej (yt)

Viterbi Table:

	Word 1 Probability of word given tag+Hidden probability at first position. P(V1)	Word 2	Word 3	Word n (Total n words)
Tag 1	P(W1|S1) + P(S1)			
Tag 2	P(W1|S2) + P(S2)			
Tag 3	P(W1|S3) + P(S3)			
Tag 4	P(W1|S4) + P(S4)			
Tag 5	P(W1|S5) + P(S5)			
Tag n	P(W1|Sn) + P(Sn)			
(Total 12 items)				

Similarly Backtrack Table is built and initialized with tag and words acting as a key and cell values as Null.

Backtracking Table:

	Word 1 Probability of word given tag+Hidden probability at first position. P(V1)	Word 2	Word 3	Word n (Total n words)
Tag 1	0			
Tag 2	0			
Tag 3	0			
Tag 4	0			
Tag 5	0			
Tag n	0			
(Total 12 items)				

Step 2:

Now we start filling the table from the second column i.e. for t > 1: vj (t) = max i∈{n,v} [vi(t − 1)pij] ej (yt)

We run two loops and calculate the emission probability of the word if it is present in the trained data and the maximum of the transition probability. This is done by comparing the new max with the previous max value and storing the greater one. The sum of the two probabilities is stored in the corresponding cell of the Viterbi table while the corresponding tag is stored in the backtrack table.

Viterbi table-

	Word 1 Probability of word given tag+Hidden probability at first position. P(V1i){i->tags}	Word 2 Max(V(t-1) x transition from t to t+1)	Word 3 Max(V(t-1) x transition from t to t+1)	Word n (Total n words) Max(V(t-1) x transition from t to t+1)
Tag 1	P(W1|S1) + P(S1)	Max(P(V1i)*P(S2i|S1i))*P(W2|S2)		
Tag 2	P(W1|S2) + P(S2)			
Tag 3	P(W1|S3) + P(S3)			
Tag 4	P(W1|S4) + P(S4)			
Tag 5	P(W1|S5) + P(S5)			
Tag n	P(W1|Sn) + P(Sn)			
(Total 12 items)				

Backtracking table-

	Word 1 Probability of word given tag+Hidden probability at first position. P(V1i){i->tags}	Word 2 Max(V(t-1) x transition from t to t+1)	Word 3 Max(V(t-1) x transition from t to t+1)	Word n (Total n words) Max(V(t-1) x transition from t to t+1)
Tag 1	0	argMax(P(V1i)*P(S2i|S1i))*P(W2|S2)		
Tag 2	0			
Tag 3	0			
Tag 4	0			

Tag 5	0			
Tag n	0			
(Total 12 items)				

Step 3:

The most likely tag with maximum probability is returned by backtracking in the “backtrack “ dataframe.

	Word 1	Word 2	Word 3	Word n
Tag 1	**ADJ**	DET	VERB	NOUN
Tag 2	PRON	NOUN	ADP	DET
Tag 3	ADJ	DET	VERB	NOUN
Tag 4	DET	**NOUN**	ADJ	PRON
Tag 5	NOUN	DET	VERB	**NOUN**
Tag n	NOUN	VERB	**ADJ**	PRON
(Total 12 items)				

We iterate over the words of the sentences from last to the beginning and compute the maximum probability of the tag associated with that word. This is appended in the tag_return list. This list denotes the tags associated with corresponding words in that order.

MCMC (complex) - def complex_mcmc()

For a given sentence a random sequence of tags is initialized, from this sequence for each position, starting from first tag, the values are changed to each tag and the corresponding joint probability is calculated using 'joint_probability' function keeping all the other tags fixed. Then a random tag is fixed at the position using the function random.choice, then same step is performed for all the tags in the tag sequence, this way one sample is generated. Number of iterations and burning iterations(samples to discard initially) are chosen, this creates number of samples from the given probability distribution. To find out the sequence from the samples, we have chosen the tag appearing the most number of times at each position.