EXPLANATION A3

def train()

Function for implementing training and computing probabilities.

Variable description:

letterAll_uniq: Contains all the unique characters

ps1: Initial Probability

This is probability of each character occurring at first position. It is the frequency that each character 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(character at first position)=Total occurrences of the character at the first position/Total number of characters at first position.

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

ps: State Probability

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

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

ps=log(P(character))

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

This is probability of two characters occurring in sequence. It is frequency of all character pairs in sequence divided by total number of character pairs. The values are stored in log. pss variable is a dictionary which contain pair of characters as pairs e.g.P[(character1, character2)]. 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(character $_{-1}$, character)=Total occurrences of (character $_{-1}$, character)/Sum of the total occurrences of all the pairs of character

pss=log(P(character $_{-1}$, character))

Emmsion:

This probability is calculated by comparing every pixel of the test image with the train image.
prob = (1-m)^p*(m)^(350-p)

Where p is the number of matched pixels and m is the noisy pixel, we assume this value to be around (0.1 to 0.2)

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 (image of the letter) and hidden variables (letter), for each test image character and corresponding train image character, the posterior probability of each character is given by P(letter| image) = (P(image|letter) * P(letter))/P(image)
We can ignore the denominator as we have to compare the score for each tag and consider the maximum. i.e. P(letter| image) = (P(image|letter) * P(letter))
This is for one image and corresponding letter, for a given sequence, we have to multiply the posterior probabilities of all the word-tag combination P(letter1| image1)*P(letter2| image2)…P(nletter2| imagen) α (P(image1| letter1) * P(letter1)) * (P(image2| letter2) * P(2)) ….. (P(imagen| lettern) * P(lettern))
Also, since the probabilities are stored in logs, we convert the above equation
to the following by taking logs on both sides
In the program the probability P(letter1| letter) 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(letter1)*P(image1|letter1)]* [P(image2| letter2)*P(letter2| letter1)]….*[P(imagen| lettern)*P(lettern| letter1n-1)]
The transition probabilities (e.g. P(letter2| letter1)) are stored in pss

.

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(image|letter).

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

_score=Hidden probability of the letter+Emission probability

= P(letter) + emmision

= ps+ emmision

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

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 72 unique tags.

dict: Dictionary of 72 letter with each letter acting as a key.

emmsion: emmsion probability

ps1: Probability that the given letter is the first letter of the sentence.

Explanation of Viterbi given in part A