README

Task 1) Design and implement Hidden Markov Model (HMM) based Part-of-Speech (POS) tagger implementing Viterbi algorithm.

Code implementation:- **# The following description is an overview of the Implementation, please have a look at the comments in the code for exact details.**

- Code is divided into 3 sections:-
  - class HMM_POS_Tagging **# Line (10 - 398)**
  - class Three_fold_cross_vali **# Line (398 - 664)**
  - Driver code **# Line (664 - 692)**

1) class HMM_POS_Tagging :-

Tasks done by this class:-

1 - Training - This method extracts all the frequencies and counts and stores them in the appropriate fields. Exact what is being counted has been mentioned in the code. Once all the data has been extracted from the training set this method calls make_emmision_matrix(), make_transition_matrix() make_transition_matrix_2(). Completing the whole training process.
Implemented in - train()
Line - (65 - 166)
Assumptions -

- All the words without tags are ignored
- All the sentences with less than 3 words/space separated characters are ignored
- No tags are reduced, everything after "_" in a unit is considered as a tag

2 - Making emmision matrix - This is the probability matrix which keeps track of all the probabilities for each unique word being associated with any unique tag.
Smoothing used: Add-1 or Laplace's smoothing
Line - (169 - 178)
Implemented in - make_emmision_matrix()

3 - Making 2nd order Transition Matrix - This is the transition matrix which keeps track of all the probabilities of all possible pairs of transitions i.e. probability of tag2 coming after tag1 in a pair of (tag1, tag2).
Smoothing used: Add-1 or Laplace's smoothing
Line - (181 - 190)
Implemented in - make_transition_matrix()

4 - Making 3rd order Transition Matrix - This is the transition matrix which keeps track of all the probabilities of all possible triplets of transitions i.e. probability of tag3 coming after tag1 and tag2 in a triplet of (tag1, tag2, tag3).
Smoothing used: Add-1 or Laplace's smoothing
Line - (193 - 207)
Implemented in - make_transition_matrix_2()

5 - 2nd order Viterbi algorithm - This is the algorithm used to decode the best order of part of speech tags from the given sequence of words. This implementation takes the previous state as context to predict the probability of the current state.
Smoothing used: Add-1 or Laplace's smoothing

Implemented in - viterbi_bigram()
Line - (210 - 291)
Assumptions -

- No transition probability has been associated for the first word of the sentences, i.e. the probability of the first word w1 being tagged as token t1 is equal to the emission probability of the pair (w1_t1)

6 - 3rd order Viterbi algorithm - This is the algorithm used to decode the best order of part of speech tags from the given sequence of words. This implementation takes the previous two states as context to predict the probability of the current state.
Smoothing used: Add-1 or Laplace's smoothing
Implemented in - viterbi_bigram_2()
Line - (294 - 396)
Assumptions -

- No transition probability has been associated for the first word of the sentences, i.e. the probability of the first word w1 being tagged as token t1 is equal to the emission probability of the pair (w1_t1)
- For the second word of the sentence previous word has been taken as the context and the probability of the second word is evaluated as P(t2/t1)*P(W/t2) where we are saying that the word is W which is tagged as t2 and the previous state is t1.

2) class Three_fold_cross_vali:

Tasks done by this class:-
1 - 3 fold cross validation- This method divides the complete sentence dataset into 3 equal sets and then evaluates the HMM model by taking all the 3 sets one by one as a test set and the other two as training sets for each iteration. This validation evaluates the Precision, Recall, Accuracy, F1-score and confusion matrix for all the folds.
Line - (425 - 661)
Implemented in - validation()

Implementation -

- All 50K sentences are shuffled using random library
- “for fold in range(3)” is applied where in the ith iteration the ith (1/3)rd set is chosen for test set and rest is set as the training set
- A new HMM_Pos_Tagging classifier is declared for each iteration and predictions of test sets are stored and then compared with original tags to get the **precisions**, **recalls f1_scores**, **accuracy** and **Tags most frequently tagged incorrectly.**

3) Driver code

In this section of code the file is read and the list of sentences is directly sent to the **Three_fold_cross_vali** class.

Task 2) Which word types are most frequently tagged incorrectly by the HMM, and why?

Answer - As you would be able to see in the output screenshot that the top 5 tags to be most incorrectly tagged are [“NN”, “NNS”, “IN”, “JJ”, “.”].
This information was extracted by getting the frequencies of incorrectly predicted tag for each actual tag and then sorting on the basis of this frequency and accordingly the top 5 most incorrectly tagged tags were found.

Reason for this incorrect tagging could be because there are several words which can be tagged for example as “NN” as well as “NNS” or maybe any other tag. Therefore due to the presence of these kinds of words which take multiple tags, many incorrect predictions are visible. As we know “NN” is a pretty common tag and since we also have subcategories of “NN” in our tag database for example “NNS” there can be words which are suitable for both “NN” and “NNS” leading the model to sometimes make incorrect predictions.