HW2: NLP

Aditi Pandurang Hande

Code Explanation:

Task 1: Vocabulary Creation

I am reading train data in df and setting the column headers as required for my logic. word_count is storing the frequency of a word in df mapped to the word, we are doing this by iterating df once and incrementing the mapped position when we encounter that word.

What is the selected threshold for unknown word replacement?
Ans: 3

What is the total size of your Vocabulary
Ans: `13751` where all unk are considered as one word.

What are the total occurrences of the special token '< unk >'after replacement?
Ans: `42044`

I have set the frequency threshold as 3 and any word with freq strictly greater than it is only considered, rest are set as <unk>. I have made unkwords where all words below threshold are stored and the df is later updated to replace them as <unk>.

Later the word_count is sorted as we need desc order in vocab.txt. The data is written in vocab using file operations. Since we wanted <unk> in the start I have explicitly written that and then written the sorted word_count where the word's freq is above threshold. (since rest are considered under <unk> and not part of vocab. As we wanted tab symbol separated, added '\\t' in between values while writing.

Task 2: Model Learning

To find transition probabilities between states and emission between state and word, used the formula given in pdf.
Transition and emission store the pair for what two entity and the value of probability. This is done by numerator/denominator as per formula for each entry in Num_transition .
Deno stores the count of each postag by total number of rows, which is count(s) in formula i.e. the denominator in both.

Num_transition stores count of occurrences of combinations of states (in correct sequence) in the train data. (essentially numerator)

Num_emission stores count of occurrences of combinations of state and word (in correct sequence) in the train data. (essentially numerator)

Used json.dump to write json in hmm.json

How many transition and emission parameters in your HMM?
Ans: transition :2025 emission: 618795…will be same for all if preprocessing for tags is not done

Task 3: Greedy Decoding withHMM

What is the accuracy on the dev data?
Ans: `92.69549511262218`

Initial stores probability of state given no prev state, this will be used to calculate probabilities for the first word of each sentence.

Approach for greedy is such that for each word we locally decide the optimal tag probability and assign that tag to word.

For every word in dev, ihv verified if that is present in train if not considered it unk. Then for that word ihv calculated probabilities for each tag. In data whenever an index is 1 new sentence is starting. Ihv used that to identify the first word of the sentence as it will use initial instead of transmission prob since it has no prev state. Formula for probability calculation is transition of last state to current * emission of current to current word. Indexlist stores the indices for all tags and then max of it is assigned used in tag_predicted which contains the mapped tag to it.

Later calculated probability as correct/total observations * 100.

Task 4: Viterbi Decoding withHMM

Accuracy on dev:`91.87435492684112`

Reading the dev data in dfdev and settingheaders for better retrieval. “`prev_statelist`
” is the datastructure that stores the indices of the assigned tags for all the previous words (this will be later used to backtrack the sentence’s optimal path, hence giving us the global maxima.

“History” stores the prev word’s probabilities that we need to calculate next states probabilities in viterbi. As we dont need values for all states I overwrite it for each word, to just store the last version. Viterbi_final_taglist stores the final ordered tags.

For every word in dev, ihv verified if that is present in train if not considered it unk. Stored its index in curidx this is used later to access emission prob values.

If the word is not . that is end of sentence and its index is one then its the first word so we cal th epob by initial * transition. And in the memoized statelist we push -1 list as no past states. And store probability in history.

Then in else i.e every other word of sentence
We calculate probability by history * emission *transition, in this case we find which states yield max and then store that probability and state.

As we encounter . all we have to do is backtrack and clear all the memoized DS as the new sentence will use fresh context..

To backtrack for final word ihv seen the max value of probability and then its previous state. And then thats states previous state and so on thill we reach first word. Now this list of tags while backtracking will be in reverse order as that of words so ihv reversed it and appended it to the final ans list.

To print the value in .out file later ihv flattened the final ans list and iterated through it to write value in files.