HW 3 - Report

MRINAL KADAM

USC ID: 3135945534

Answers to the questions in the assignment:

- Selected threshold for unknown words replacement is **2** i.e., any word with a **frequency of lesser than 2** will be considered as an **unknown word**.

- Total size of my vocabulary: **21722**
- Total occurrences of the special token '< unk >' after replacement: **17952**
- Total occurrences of the special token '< num >' after replacement: **21241**

- Transition parameters in my HMM: **2116**
- Emission parameters in my HMM: **28831**

- Accuracy of **greedy** algorithm on dev data: **93.69 %**
- Accuracy of **Viterbi** algorithm on dev data: **94.95 %**

Brief explanations about my solution:

Task 1: Vocabulary Creation

- I read the training file and stored it into a data frame, separating each different sentence in the input file (separated by a blank space) into a different row of the data frame.
- Data pre-processing:
    1. checked for words that are entirely numbers and replaced them with '<num>' tag using regex
    2. checked for words that are unknown (have a frequency threshold lesser than 2) and replaced them with '<unk>' tag (this was done after I made a dictionary containing the (key:value) pair as (word:word_count) for all the words present in my training data frame)
- I then calculated the sum of all the words that have been either tagged as <num> or <unk> and stored their tags and counts at the top of the 'vocab.txt' file, followed by the rest of the words and their counts in descending order as asked.

Task 2: Model Learning

- I calculated the transition and emission probabilities for my training vocab by using the following formulae:

$$t(s'|s) = \frac{\text{count}(s \to s')}{\text{count}(s)}$$
$$e(x|s) = \frac{\text{count}(s \to x)}{\text{count}(s)}$$

where $t(\cdot|\cdot)$ is the transition parameter and $e(\cdot|\cdot)$ is the emission parameter.

t(s'|s) – s': current tag, s: previous tag – transition from previous tag to the current tag

e(x|s) – x: current word, s: it's tag – emission of current word given its tag

I also applied Laplace/Additive smoothing on the transition probabilities by considering alpha = 1. So, my transition probabilities that were previously zero, got converted to very small non-zero values with the help of which, I could get a slightly better accuracy.

I stored these separate dictionaries into a high-level dictionary with 'transition' and 'emission' as the keys and output it as 'hmm.json' as asked.

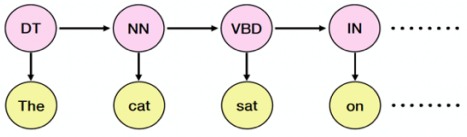
- I used these probabilities extracted from my training vocab for training my greedy and Viterbi models.

Task 3: Greedy Decoding with HMM

- I performed pre-processing (replacing with <num> and <unk> tags) on dev and test data.
- I then passed this data through my greedy decoding algorithm –

  For every word in every row of my data frame (formed such that each different sentence is a different row of my data frame), I iterated through all possible (45) tags and calculated the product of my emission and transition probabilities by looking them up in the dictionaries created before, according to the below formula:

Greedy Decoding



$$s_j^* = \arg\max_{s_j} t(s_j | s_{j-1}^*) e(x_j | s_j), \quad \forall j$$

At the end, the tag that gave me the highest product became my predicted POS tag for the word under consideration.

- I checked my accuracy on dev data as the true tags were also given. I tried several different permutations and combinations of my parameters like different values of my threshold value for <unk>, making my word model case sensitive/insensitive and trying out smoothing to increase my accuracy. Finally, I considered my threshold value as 2, made my model case sensitive and performed Laplace smoothing on the transition probabilities to get as high of an accuracy as I could.

For test data, I used these final parameters and predicted the POS tags for the given words and stored it in 'greedy.out' file as asked.

Task 4: Viterbi Decoding with HMM

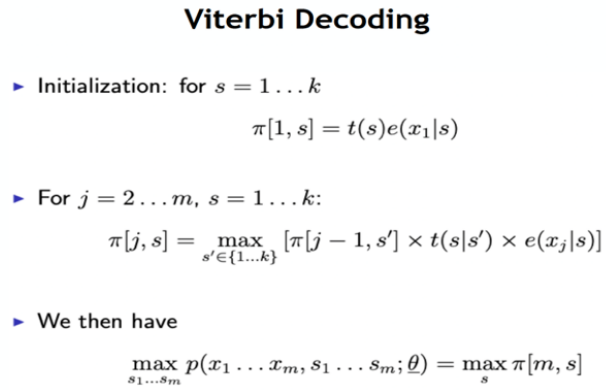
- I used the pre-processed (replaced with <num> and <unk> tags) dev and test data.
- I then passed this data through my Viterbi decoding algorithm –

  I went through every row of my data frame and for the first word in every row i.e. for the beginning word of each sentence, I calculated the product of the emission and transition(s|<s>) probabilities by iterating over all tags. Alongside, in my data structure, I also stored the previous tag which was <s> in this case.

  For all the other words in every row, I iterated over all the tags and calculated the product of the emission and transition probabilities with the product obtained for all the possible tags of the previous word, that had been stored in my data structure. I stored only the max values for a particular tag along with that tag in my data structure.

  After doing this for all rows over all words over all tags, I backtracked on my data structure and picked out the tags that gave me the highest products for each word and these became my predicted POS tags.

  The following reference was used for coding this algorithm:



Viterbi Decoding

- Initialization: for $s = 1 \dots k$

$$\pi[1, s] = t(s)e(x_1|s)$$

- For $j = 2 \dots m$, $s = 1 \dots k$:

$$\pi[j, s] = \max_{s' \in \{1 \dots k\}} [\pi[j-1, s'] \times t(s|s') \times e(x_j|s)]$$

- We then have

$$\max_{s_1 \dots s_m} p(x_1 \dots x_m, s_1 \dots s_m; \underline{\theta}) = \max_s \pi[m, s]$$

- I checked my accuracy on dev data as the true tags were also given.

  For test data, I used the final parameters and predicted the POS tags for the given words and stored it in ‘viterbi.out’ file as asked.