CSCI 544 – Applied Natural Language Processing

Homework Assignment No 2 – Submission

- Shubham Sanjay Darekar

Submission details

The submission contains the following –

1. The executable python file containing code of this assignment.
2. Readme file explaining the execution steps for the python file.
3. Zipped `Out` folder, containing vocab.txt, hmm.json, greedy.json and viterbi.json.
4. This PDF file containing the explanation to implementation and answers to the asked questions.

Highest Accuracy Details on dev dataset

I have observed maximum accuracy with following parameters –

Parameter	Value	Comment
Threshold	2	Any word with less than 2 occurrences in train will be considered as unknown word
use_pseudo_words	True	Replacing the words below the threshold in train dataset and unknown words in dev/test dataset with pseudo words. More details in next section on pseudo words
use_log_likelihood	True	Using log likelihood to decode the algorithm

Greedy – 94.23 %

```
What is the accuracy on the dev data?
--->>>>> Greedy dev Accuracy: 94.23 %
```

Viterbi – 95.20 %

```
What is the accuracy on the dev data?
--->>>>> Viterbi Accuracy: 95.2 %
```

Answers to inline questions

The answers are also printed by the python executable file. These values are obtained with the above parameters.

1. *What threshold value did you choose for identifying unknown words for replacement?*

   Ans: Threshold – 2. Any word with occurrence less than 2 are replaced with pseudo words and unknown tags. For details check the explanation below.

2. *What is the overall size of your vocabulary?*

Ans: Size - 23190 (Includes 8 pseudo word tokens)

3. *How many times does the special token "< unk >" occur following the replacement process?*
   Ans: I have updated the words with frequency below threshold mentioned above with pseudo words with the following logic

```python
def psudo_words_convert(word:str, use_pseudo_words: bool):
    """
    Converts the unknown words into pseudo words or assigns <unk> tag
    ## Reference - http://www.cs.columbia.edu/~mcollins/hmms-spring2013.pdf

    Inputs -
    word:str                : Input word
    use_pseudo_words: bool  : Flag stating if pseudo word implemtion should be considered

    Outputs -
    string  :   Pseudo word or <unk> tag

    """
    if not use_pseudo_words:
        return '<unk>'
    else:
        if re.match(r'^[A-Z][a-z]*$',word):
            return 'init_cap'
        elif re.match(r'^[0-9]{4}$',word):
            return 'four_digits'
        elif re.match(r'^.*\.$',word):
            return 'words_ending_with_period'
        elif re.match(r'^[A-Z]*$',word):
            return 'all_caps'
        elif re.match(r'^[a-z]*$',word):
            return 'lowercase'
        elif re.match(r'^[0-9]*\.[0-9]*$',word):
            return 'number_and_period'
        elif re.match(r'^[0-9]*\,[0-9]*\.[0-9]*$',word):
            return 'number_and_period_and_comma'
        else:
            return '<unk>'
```

Reference - http://www.cs.columbia.edu/~mcollins/hmms-spring2013.pdf

```
How many times does the special token "< unk >" occur following the replacement process?
---->>>>>
Following tokens/ pseudo words were used in replacement process. The corresponding occurances are mentioned
<unk> :4660
init_cap :5366
four_digits :41
words_ending_with_period :76
all_caps :663
lowercase :7396
number_and_period :1803
number_and_period_and_comma :6

Total replaced: 20011
```

Frequency count:
<unk> :4660
init_cap :5366
four_digits :41
words_ending_with_period :76
all_caps :663
lowercase :7396
number_and_period :1803
number_and_period_and_comma :6

Total replaced: 20011

4. *How many transition and emission parameters in your HMM?*

   Ans:

```
How many transition and emission parameters in your HMM?
--->>>>>
Initial_pi : 41
Emissions : 30372
Transition : 1375
```

Initial_pi: **41**
Emissions: **30372**
Transition: **1375**

Ideally, initial_pi should have length of number of tags, Emissions should be (tags x words in vocab) and Transitions should be (tags x tags), but as I am not considering the value with 0 probability, the numbers are less than the ideal values.

5. *For Greedy decoding, what is the accuracy on the dev data?*

   Ans: The accuracy is - **94.23 %**

```
What is the accuracy on the dev data?
--->>>>> Greedy dev Accuracy: 94.23 %
```

6. *For Viterbi decoding, what is the accuracy on the dev data?*

   Ans: The accuracy is – **95.20 %**

```
What is the accuracy on the dev data?
--->>>>> Viterbi Accuracy: 95.2 %
```

More details on implementation

The code contains comments explaining all the steps performed, the below information is an addition to those comments

1. *Parameters of Algorithm -*

```
threshold_unknown = 2           ## Threshold, any word with less than N occurrences in train will be considered as unknown word
is_dev_run = True               ## Flag to check the accuracy on dev dataset
use_pseudo_words = True         ## Flag to use the pseudo words function to convert unknown words
init_value = 0                  ## Setting initial value of vocab word initialization
use_log_likelihood = True       ## Set if you need to use log likelihood while calculating probabilities
eps = 1e-300                    ## using small epsilon value for log likelihood
```

2. *Task 1 – Vocabulary Creation*
    - In creation of vocabulary, I have iterated through the train dataset and counted the occurrences of each word adding them to a dictionary.
    - All the values below specified threshold are replaced using the pseudo_word_convert function mentioned above.
    - The final output is then written to vocab.txt file.

    Function –

```
def vocab_creation(train_data: list, vocab_path:str, use_pseudo_words:bool, threshold:int = 3, init_value:int = 0):
    """
    Reads and iterate through the train dataset and outputs vocabulary to vocab.txt
    Inputs -
        train_data:list         : Parsed json object of train data with keys labels and sentences
        vocab_path:str          : absolute path to output file (txt file)
        use_pseudo_words:bool   : Flag stating if pseudo word implemtion should be considered
        threshold:int           : Any word with less than N occurances in train will be considered as unknown word (optional, def: 3)
        init_value:int          : Initial value of vocab word initialization used for smoothening (optional, def: 0)
    Output -
        dict_vocab_sentences    : Dictionary containing unique words and occurance count
        dict_vocab_pos          : Dictionary containing unique pos tags and occurance count
        train_data              : Updated train dataset using the parameters defined (pesudowords and thresholds)
    """
```

    Note: I have created a parameter called init_value, which assigns a default frequency value to all the words in attempt of smoothening. But I have observed better accuracies with its value set to 0.

3. *Task 2 – Model Learning*

    The model learning function created the initial pi, transition, and emission dictionaries.

Dictionary Name	**Key Format**
Initial_pi	Pos_tag
Transition	(Pos_tag1, Pos_tag2)
Emission	(Pos_tag, vocab_word)

    The final dictionaries are output to hmm.json file.

    Function details –

```
def model_learning(train_data:list, dict_vocab_pos:dict, hmm_path:str):
    """
    Method to calculate initial probabilities, emission probabilities and transition probabilities
    Writes the dictionaries to hmm.json file.
    Inputs -
    train_data:list         : Processed training dataset
    dict_vocab_pos:dict     : list of pos tags and their occurances
    hmm_path:str            : Path to output json file

    Outputs -
    emission        : Emission probabilities - key - (tag, word)
    transition      : Transition probabilites - key - (tag, next_tag)
    initial_pi      : Initial probabilities - key - (tag)
    """
```

4. *Task 3: Greedy Decoding with HMM*
    - This task used the dictionaries from previous task to perform greedy decoding.
    - As pseudowords implementation is used, if any out of vocabulary word occurs while decoding, it is either converted to a pseudo word, if it matches any of the regular expressions, or an unknown tag <unk> is assigned to the that word. Then the emission probabilities are considered with the replacement.
    - Log likelihood is being calculated for each of the emission transition pairs. The goal is to maximize the log likelihood in each step.
    - To avoid math errors, a small step value epsilon is added to all the probabilities. In my case, Epsilon = 1e-300
    - This tasks generates labels for test dataset and outputs them to greedy.json

    Function ->

```
def greedy_hmm_decoding(emission: dict, transition: dict, initial_pi: dict, dict_vocab_sentences:dict,test_data: list,dict_vocab_pos: di
    """
    Greedy decoding algorithm.
    Writes the greedy.json file with predicted labels

    Inputs -
        emission: dict              : Emission probabilities - key - (tag, word)
        transition: dict            : Transition probabilites - key - (tag, next_tag)
        initial_pi: dict            : Initial probabilities - key - (tag)
        dict_vocab_sentences:dict   : Dictionary containing unique words and occurance count
        test_data: list             : Processed training dataset
        dict_vocab_pos: dict        : list of pos tags and their occurances
        greedy_out_path:str         : Path to greedy.json output file
        use_pseudo_words:bool       : Flag stating if pseudo word implemtion should be considered
        use_log_likelihood:bool     : Flag stating if log likelihood implemtion should be considered
        eps:float                   : Small number added to avoid math error in log

    Outputs -
        test_data       : predicted labeled test dataset

    """
```

5. *Task 4: Viterbi Decoding with HMM*
    - Like greedy task, Viterbi decoding also uses the dictionaries created in model learning.
    - The pseudo words replacement and log likelihood implementation is similar to the Greedy algorithm

- At each pass, two matrices of shape (number of words x total POS tags) are initialized with 0s. One of them is used to store the Viterbi probabilities and other stores the information required to backtrack the path of maximum probability.
  Function:

```
def viterbi_hmm_decoding(emission:dict ,transition:dict , initial_pi: dict, dict_vocab_sentences:dict, test_
    """
    Viterbi decoding algorithm.
    Writes the viterbi.json file with predicted labels

    Input -
        emission: dict              : Emission probabilities - key - (tag, word)
        transition: dict            : Transition probabilites - key - (tag, next_tag)
        initial_pi: dict            : Initial probabilities - key - (tag)
        dict_vocab_sentences:dict   : Dictionary containing unique words and occurance count
        test_data: list             : Processed training dataset
        dict_vocab_pos:dict         : list of pos tags and their occurances
        viterbi_out_path:str        : path to viterbi.json output file
        use_pseudo_words:bool       : Flag stating if pseudo word implemtion should be considered
        use_log_likelihood:bool     : Flag stating if log likelihood implemtion should be considered
        eps:float                   : Small number added to avoid math error in log
    Output
        test_data       : predicted labeled test dataset
    """
```

Note: The code is written dynamically to change according to the set parameters. For example, if the use_log_likelihood flag is set to false, the decoding is done without taking the log. This was done for obtaining the maximum accuracy.

Pseudo Words replacement details

Ref - http://www.cs.columbia.edu/~mcollins/hmms-spring2013.pdf

The low frequency words are replaced by following self-explanatory tags

1. '<unk>'
2. 'init_cap'
3. 'four_digits'
4. 'words_ending_with_period'
5. 'all_caps'
6. 'lowercase'
7. 'number_and_period'
8. 'number_and_period_and_comma'

The unknown <unk> tag is assigned if none of the other tags are applicable to the word. This helps in understanding the behavior of out of vocabulary words.

Accuracies obtained with different parameters on dev dataset.

Parameter	Greedy Accuracy	Viterbi Accuracy
Threshold = 1 or 0 use_pseudo_words = True use_log_likelihood = True	93.53 %	94.908 %

***** Highest** Threshold = 2 use_pseudo_words = True use_log_likelihood = True	94.23 %	95.2 %
Threshold = 3 use_pseudo_words = True use_log_likelihood = True	93.914 %	94.905 %
Threshold = 4 use_pseudo_words = True use_log_likelihood = True	93.681 %	94.688 %
Threshold = 5 use_pseudo_words = True use_log_likelihood = True	93.434 %	94.455 %
Threshold = 0 or 1 use_pseudo_words = False use_log_likelihood = False	92.952 %	56.197 %
Threshold = 2 use_pseudo_words = False use_log_likelihood = False	93.507 %	94.771 %
Threshold = 3 use_pseudo_words = False use_log_likelihood = False	92.99 %	93.371 %
Threshold = 4 use_pseudo_words = False use_log_likelihood = False	92.635 %	94.034 %
Threshold = 5 use_pseudo_words = False use_log_likelihood = False	92.24 %	93.736 %

Code Execution details

Code execution details are given in the readme.txt file.

Reference Links

The links have been added to the code as well in comments

- Pseudo Words Implementation - http://www.cs.columbia.edu/~mcollins/hmms-spring2013.pdf
- Fastest method to sort dictionary - https://stackoverflow.com/questions/613183/how-do-i-sort-a-dictionary-by-value
- Finding maximum index - https://stackoverflow.com/questions/11530799/python-finding-index-of-maximum-in-list