Natural Language Processing [CS4120]

Instructor : Professor Lu Wang
Assignment 1

Deadline: February 5th, 2020 at 11:59 PM on Blackboard

For the programming questions, you can use Python (preferred), Java, or C/C++. Please include a README file with detailed instructions on how to run your code. Failure to provide a README file will result in **deduction of points** (5 to 10 points per problem). Your final deliverable for this homework should consist of one zipped folder with i) text file(s) with your answers for questions requiring textual answers, ii) zipped folders with code (one folder per problem), and iii) README describing how to run your code for each of the programming problem.

This assignment has to be done individually. If you discuss the solution with others, you should indicate their names in your submission. If you use ideas/code from any online forum, you should cite them in your solutions. **Violation of the academic integrity policy is strictly prohibited, and any plausible case will be reported once found. No exceptions will be made.**

1 Naive Bayes for Text Categorization [10 points]

You are a NLP engineer at the New York Times and have been assigned a task to classify news articles into appropriate categories. You are given the top 4 most frequent keywords in each of the articles below, and each news article is labeled with a category - Business or Politics (class):

1. zuckerberg, war, senate, trump : **Politics**
2. trump, markets, billionaire, zuckerberg : **Business**
3. billionaire, problems, elections, trump : **Politics**
4. war, zuckerberg, internet, problems : **Business**
5. war, elections, problems, trade : **Politics**
6. trade, investors, billionaire, ceo : **Business**
7. ceo, trump, war, internet : **Politics**
8. billionaire, trade, internet, ceo : **Business**

And test articles:

1. A1 : zuckerberg, ceo, internet, billionaire
2. A2 : war, trump, problems, elections

Your task is to compute the most likely class for articles A1 and A2. You will start with building a Naive Bayes classifier and using add-λ smoothing (with $\lambda = 0.1$). For this question, **show your work** of computing prior, conditional, and posterior probabilities. Do not write/submit code for this question.

2 Word Sense Disambiguation [15 points]

Given the following sentences:

1. The plane was about to $bank_1$ as if it was returning to the airport.
2. Shawn deposited his salary money straight into his $bank_2$ at the start of the month.
3. For the past 10 or so years the $bank_3$ of the river has been neglected by the authorities.
4. John is the current manager of the $bank_2$ in city of Boston.
5. There are beautiful trees along the $bank_3$ of the river.
6. Raj decided not to $bank_4$ with the old family banks.

The aforementioned sentences show four different contexts in which the word "bank" can be used, marked as 1, 2, 3 and 4. As discussed in class, collocational features, the features at specific positions near target word, can be used to train a supervised learning model, such as Naive Bayes, to perform word sense disambiguation. Considering the aforementioned sentences as the known corpus, answer the question below.

Find the collocation features from a window of two words to the right and the left of the word "bank". Prepare to present the features in terms of the words and their respective part-of-speeches for each sentence. Format can be : [$w_{i-2}, POS_{i-2}, w_{i-1}, POS_{i-1}, w_{i+1}, POS_{i+1}, w_{i+2}, POS_{i+2}$], where i is the index of word "bank" in a given sentence. For the POS tags, you can refer to the textbook or use using any off-the-shelf POS tagger (please indicate which tagger you used in the submission).

No need to write code, show the answer directly.

3 Language Modeling [40 points]

Your task is to train trigram *word-level* language models from the following English training data and then test the models.
The training data can be downloaded at:

http://bit.ly/3a2nzBw

For all questions, please submit both the code and output.

Overall data preprocessing instructions:

1. Remove blank lines from each file.
2. Remove newline characters.
3. Remove duplicate spaces.
4. Replace words in training set that appear ≤ 3 times as "UNK".
5. Do **not** remove punctuation

3.1 [5 points]

Across all files in the directory (counted together), report the unigram, bigram, and trigram counts. Submit these counts in 3 separate files named **unigramCounts.txt, bigramCounts.txt and trigramCounts.txt**.

Note: You can use any word tokenizer to tokenize the dataset e.g. nltk word_ tokenize, although for creating the n-grams do **not** use any libraries.

3.2 [15 points]

For the given test dataset:
http://bit.ly/3a9Szjg
Calculate the perplexity for each file in the test set using linear interpolation smoothing method. For determining the λs for linear interpolation, you can divide the training data into a new training set (80%) and a held-out set (20%) , then using grid search method (you need to first decide a set of values for λs).

1. First, report all the candidate lambdas used for grid search and the corresponding perplexities you got on the held-out set
2. Report the best λs chosen from the grid search, and explain why it's chosen (i.e. leveraging the perplexities achieved on the held-out set).
3. Report the perplexity for each file in the test set (use the best λs obtained from grid search to calculate perplexity).

Submit these perplexities and your report in a file named ***perplexitiesInterpolation.txt***.

3.3 [15 points]

Build another language model with add-λ smoothing. Use $\lambda = 0.1$, $\lambda = 0.2$ and $\lambda = 0.3$. Report the perplexity for each file in the test set (for all 3 λ values). Submit these perplexities and your report in a file named ***perplexitiesAddLambda.txt***.

3.4 [5 points]

Based on your observation from above questions, compare linear interpolation and add-lambda smoothing by listing out their pros and cons.

4 POS Tagging with HMM and Sentence Generation [35 points]

The training dataset is a subset of the Brown corpus, where each file contains sentences in the form of tokenized words followed by POS tags. Each line contains one sentence. Training dataset can be downloaded from here: https://bit.ly/2T9nRAA. The test dataset (which is another subset of the Brown corpus, containing tokenized words but no tags) can be downloaded from here: https://bit.ly/2uBQFaI. Information regarding the categories of the dataset can be found at: https://bit.ly/2QGZj06.

Your task is to implement a part-of-speech tagger using a bi-gram HMM. Given an observation sequence of n words w_1^n, choose the most probable sequence of POS tags t_1^n. For the questions below, please submit both code and output.

[Note: During training, for a word to be counted as unknown, the frequency of the word in training set should not exceed a threshold (e.g. 3). You can pick a threshold based on your algorithm design. Also, you can implement smoothing technique based on your own choice, e.g. add-α.]

4.1 [5 points]

Obtain frequency counts from the collection of all the training files (counted together). You will need the following types of frequency counts: word-tag counts, tag un-igram counts, and tag bi-gram counts. Let's denote these by $C(w_i, t_i)$, $C(t_i)$ and $C(t_{i-1}, t_i)$ respectively. Report these quantities in different output files.

4.2 [2 points]

A transition probability is the probability of a tag given its previous tag. Calculate transition probabilities of the training set using the following equation:

$$P(t_{i-1}, t_i) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

4.3 [3 points]

An emission probability is the probability of a given word being associated with a given tag. Calculate emission probabilities of the training set using the following equation:

$$P(w_i, t_i) = \frac{C(w_i, t_i)}{C(t_i)}$$

4.4 [10 points]

Generate 5 random sentences using the previously learned HMM. Output each sentence (with the POS tags) and its probability of being generated.

Hint:
With the help of emission probabilities and transition probabilities collected from 4.2 and 4.3,

1. Start with '`<start>`' tag (you can use other start of sentence tags).
2. Choose next tag based on random choice but considering probabilities
   e.g. `tag_draw = random.choices(<tags>,<tag-probabilities>,<number of tags to draw>).`
3. Now choose word for the corresponding tag using emission probabilities (all the words that can be generated from that tag and corresponding probabilities they can be generated with.)
   e.g. `word_draw = random.choices(<words>,<word-probabilities>,<number of tags to draw>).`
4. Keep repeating steps 2 and 3 till you hit end token '`</end>`' (or other end of sentence tags you use).
5. Report the sentence and the probability with which this sentence can be generated.

4.5 [15 points]

For each word in the test dataset, derive the most probable POS tag sequence using the Viterbi algorithm; pseudo-code can be found in the textbook http://web.stanford.edu/ jurafsky/slp3/ed3book.pdf under **Figure 8.5**. Viterbi algorithm should be implemented following the pseudocode provided for reference.

Hint: Traversing through back-pointer data structure at the end of algorithm would provide information about the best possible previous tag.

Submit the output in a file exactly with the following format (where each line contains no more than one pair):

```
< sentenceID = 1 >
word/tag
word/tag
```

```
....
word/tag
< EOS >
< sentenceID = 2 >
word/tag
word/tag
....
word/tag
< EOS >
```