Natural Language Processing [CS4120/CS6120]

Instructor : Professor Lu Wang
Assignment 1

Deadline: October 8th, 2019 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 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 forums, 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]

Given the following short documents, each labeled with a genre (class):

1. murder, cars, map, treasure : **Action**
2. treasure, love, conspiracy, murder : **Drama**
3. conspiracy, robbery, crash, treasure : **Action**
4. cars, murder, enemy, robbery: **Drama**
5. cars, crash, robbery, family: **Action**
6. family, vendetta, conspiracy, betrayal : **Drama**
7. betrayal, treasure, cars, enemy : **Action**
8. conspiracy, family, enemy, betrayal : **Drama**

And test documents:

1. D1 : murder, betrayal, enemy, conspiracy

2. D2 : cars, treasure, robbery, crash

Your task is to compute the most likely class for D1 and D2. You will start with building a Naive Bayes classifier and using add-λ smoothing (with $\lambda = 0.2$). 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 and Feature Selection [25 points]

2.1 Feature Construction [5 points]

Given the following sentences:

1. The company $board_1$ of directors has seven members.
2. Several loose $board_2$ creaked as I walked on them.
3. We all $board_3$ the plane for Oslo tomorrow.
4. John nailed the $board_2$ over the window.
5. Bella wanted to $board_3$ the bus to Chicago.
6. They needed to get more senators on $board_4$ for the bill to pass.

The aforementioned sentences show four different contexts in which the word "board" 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 "board". 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 "board" in a given sentence.

No need to write code, show the answer directly.

2.2 Selecting Salient Features [20 points]

In the class you saw that there are two kinds of features: i) collocational and ii) bag of words. Having seen how to extract collocational features (by hand) to disambiguate the sense for a word with multiple senses in the previous problem, now let's try to understand which features are important to disambiguate the word senses on a bigger corpus. You can use a combination of collocational and bag of words features to conduct feature selection.

Dataset: We will use the English Lexical Sample task from Senseval for this problem. The data files for this project are available here: https://bit.ly/2kKEgwx

It consists of i) a corpus file (wsd_data.xml) and ii) a dictionary (dict.xml) that describes commonly used senses for each word. Both these files are in XML format. Every lexical item in the dictionary file contains multiple sense items, and each instance in the training data is annotated with the correct sense of the target word for a given context.

The file `wsd_data.xml` contains several `<welt>` tags corresponding to each word in the corpus. Each `<welt>` tag has an attribute `item`, whose value is "word.pos", where "word" is the target word and "pos" represents the part-of-speech of the target word. Here 'n', 'v', and 'a' stand for noun, verb, and adjective, respectively.

Each `<welt>` tag has several `<instance>` tags, each corresponds to an instance for the word that corresponds to the parent `<welt>` tag. Each `<welt>` tag also has an `id` attribute and contains one or more `<ans>` and a `<context>` tag. Every `<ans>` tag has two attributes, `instance` and `senseid`. The `senseid` attribute identifies the correct sense from the dictionary for the present word in the current context. A special value "U" is used to indicate if the correct sense is unclear. You can discard such instances from your feature extraction process for this assignment (we keep these cases so that you can take a look and think about how they can be utilized as well for real-world applications).

A `<context>` tag contains:

`prev-context <head> target-word <head> next-context`

1. `prev-context` is the actual text given before the target word
2. `head` is the actual appearance of the target word. Note that it may be morphological variant of the target word. For example, the word "begin.v" could show up as "beginning" instead of "begin" (lemma).
3. `next-context` is the actual text that follows the target word.

The dictionary file simply contains a gloss field for every sense item to indicate the corresponding definition. Each gloss consists of commonly used definitions delimited by a semicolon, and may have multiple real examples wrapped by quotation marks being also delimited by a semicolon.

2.2.1 Feature extraction [10 points]

Your first task is to extract features from the aforementioned corpus. (1) Start with bag-of-word features and collocation features (define your own window size, see Hints below). (2) Design new type of features. Submit the code and output for both.

2.2.2 Feature selection [10 points]

Now, with the extracted features, perform feature selection to list top 10 features that would be most important to disambiguate a word sense. (1) Design your own feature selection algorithm and explain the intuition. (2) List the top 10 features in your answer key and also provide your code for this task. Submit the code and output for both.

You can use following resources to read ways to perform feature selection:
https://scikit-learn.org/stable/modules/feature_selection.html
https://www.datacamp.com/community/tutorials/feature-selection-python

Hints:

1. You may want to represent each instance as a feature vector. Recall, a feature vector is a numerical representation for a particular data instance. Each feature vector will have a respective target, a numerical representation of the sense that the instance was labeled with.

2. In order to work with the scikit-learn API, you will have to format all of your feature vectors into a numpy array, where each row of the array is an individual feature vector. You will have to do the same for your targets - put them into a 1-dimensional numpy array.

3. As seen in class, bag of words features are the count for each word within some window size of the head word. For this problem, you can set the window size large enough to include all of the tokens in the context. You may want to keep punctuation, numbers, etc as they can be useful?

4. As also seen in previous problem, collocational features are the n-grams that include the head word and their counts. For this problem, you can just extract the bi-grams and tri-grams that include the head word. If there are multiple head words in a single context, you should extract bi-grams and tri-grams for each one. You can represent these n-grams as a single "word" by joining the tokens in the n-gram together using underscores to form features.

3 Language Modeling [30 points]

Your task is to train trigram *word-level* language models from the following English training data and then test the models
Training data:
https://www.dropbox.com/s/puo7jygnh9m52ze/train.zip?dl=0

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

Data preprocessing instructions:

1. Remove blank lines from each file.

2. Replace 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 word-level counts. Submit these counts in a file named ***ngramCounts.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 [10 points]

For the given test dataset:
https://www.dropbox.com/s/ik98szmqbsq2wtd/test.zip?dl=0
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.

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 on test set).
4. Based on the test file's perplexities you got write a brief observation comparing the test files.

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

3.3 [10 points]

Build another language model with add-λ smoothing. Use $\lambda = 0.1$ and $\lambda = 0.3$.

1. Report the perplexity for each file in the test set (for both the λ values).
2. Based on the test file's perplexities you got write a brief observation comparing the test files.

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/2kJI0yc 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/2lMybzP Information regarding the categories of the dataset can be found at: https://bit.ly/2mhF6RT.

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. 5). 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:

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

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.
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>.'
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. So when you are at the second last word of the sentence, calling back-pointer here would give the tag information for the first word in the sentence.

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 >
```