CSCE 5543-STATISTICAL NLP HW3

Mamata Shrestha

1 Algorithms

We are interested in determining the probability of a sentence which can be useful for various tasks such as machine translation, spell correction, speech recognition etc. We can do it by employing various statistical or probabilistic techniques. We can calculate the probability of sentence or upcoming word by using following formula respectively:

$$p(W) = p(w1, w2, w3, w4, w5, ..., wn)$$

$$p(w5|w1, w2, w3, w4)$$

Using chain rule of probability,we have,

$$p(w1, w2, w3, w4) = p(w1) * p(w2|w1) * p(w3|w1, w2) * p(w4|w1, w2, w3)$$

For instance,
P(<s> I want english food </s>) = P(I|<s>) × P(want|I) × P(english|want) × P(food|english) × P(</s>|food)

Since these values are very small, we can also use log to calculate the probabilities.

Laplace Smoothing : For the bigrams that are not present in the text file, we will have probability and bigrams count as zero. Laplace smoothing helps to tackle the problem of zero probability by changing the count of bigrams from 0 to 1 and pretending that we all each word one more time than we did. So the formula is given by:

$$P^*(W_n|W_{n-1}) = \frac{C(W_{n-1}, W_n) + 1}{C(W_{n-1}) + V}$$

2 Data Structure

Hash table is used for counting the frequency of each token and bigrams in the given text file. Hash table is an abstract data structure that maps keys to a value. It uses a hash function to get an index of the hash table where the value corresponding to a key is stored. Collision occur if more than key has same hash value. Therefore, separate chaining method is used to handle the collisions. The chaining method arranges the values that are mapped to the same index in a list. Dictionary data structure is also used to store the bigrams and unigrams count after loading them from a text file.

3 Text Pre-processing

1. **Convert short form words:**
   Conversion of words like ́ve to have, i'm to I am and n't to not and so on for better tokenization in next steps.
2. **Remove punctuation:**
   Removed the punctuation to improve the tokenization and the quality of bigrams obtained.
3. **Tokenization:**
   Used nltk built-in function to tokenize the input.
4. **Text normalization:**
   All tokens are converted to its lower case so that the counting of token does not become case sensitive.

4 Running code

The input directory with the *Shakespeare.txt* file must be in the same directory with the code files. I have hardcoded the input file path in bash script. Run code by specifying the shell script, which runs code files as follows:
./run.sh

By executing bash script, first bigrams, unigrams and normalized bigrams are calculated and stored in the text file inside outputs dir by running *count_bigrams.py* script. After this, we calculate the probability of the input sentence with/without laplace smoothing based on user inputs by running *prob_sentence.py* scripts. This script takes two inputs from user: 1) input sentence for which we want to calculate the probability and 2) y/n based on whether we want to use laplace smoothing or not. And the third scripts is *generate_sentence.py* which generates the sentence based on the given input word with/without using laplace smoothing.

5 Overview

For the given task, I first calculated the bigrams, unigrams and normalized bigrams and saved them in a text file under outputs directory. For calculating the probability of sentence, I used a log so that the probability of sentence is the sum of log of conditional probability of each word given previous words. I replace the probability of bigrams which are not found in the text file with log(1) which is 0.
For the second part where we generate the sentence, I filtered the words whose bigrams are present in the bigrams hashtable while finding the list of next potential words for a given word. I used this filter to avoid the words that are not present in the bigrams else those word will result None as the next word in next step. After that, I extracted first five words, shuffle them randomly and choose the word at index zero as the next word.
In the method without laplace smoothing, if the current sentence doesń't have any bigrams then the next predicted word will be None. For instance, twitter word does not appear in the text file so it doesń't have any bigrams, the program return twitter only if we try to generate sentence using this word as the start word. But with laplace smoothing, we assume that we have seen all possible bigrams from tokens at least once and find the next word using it.

6 Outputs

While calculating the sentence probability, for the bigrams with count zero, the probability also becomes zero. But, if we use laplace smoothing, we get very small value instead of zero.
Without Laplace Smoothing
The probability of sentences was found to be:

1. All the world's a stage = -4.0631
2. to be or not to be = -14.9995
3. astronaut Internet telephone = 0.00

Also, the predicted sentence of length 8 starting with each word is given by:

1. love: love is to be a man is it
2. sleep: sleep thou shalt be not a good lady
3. twitter: twitter

I have used random shuffle for selecting next words, therefore, result might not exactly similar if program is run again.

With Laplace Smoothing
The probability of sentences was found to be:

1. All the world's a stage = -30.6675
2. to be or not to be = -23.9158
3. astronaut Internet telephone = -17.4636

Also, the predicted sentence of length 8 starting with each word is given by:

1. love: love you will not know what you will
2. sleep: sleep in my love is the gods had
3. twitter: twitter oyster dote on the world is the