BBM497: Introduction to Natural Language Processing Lab. **(Due: 30/03/2020)**

Submission Assignment #1

Instructor: Necva Bölücü *Name:* Utku İpek, *Netid:* 21627356

Abstract

In this assignment, the goal is to create a program that generate new sentences using the n-gram models and interpret the results by comparing the probability and perplexity of these new sentences. As the dataset, Facebook Children Stories is used. The data structures I've used, functions I've implemented and the interpretation of the generated sentences can be found in below sections.

1 Data Structures

My implementation contains a total of two types of data structures which are dictionary and list. The first data structure I have created is a list structure for the sentences in the dataset. After reading and processing each sentence, I have added them to the list named list_of_sentences which is a class variable of the LanguageModel class. To build the n-gram models, I have declared three dictionaries named unigrams, bigrams, and trigrams as class variables. According to n value, each of these dictionaries holds the word groups as their keys and holds the counts of these keys in the whole dataset as their values. After creating these dictionaries, I have sorted each of them using OrderedDict type from Python collections. Also, I have declared three more dictionaries as class variables for the probabilities. These dictionaries are unigram_probs, bigram_probs, and trigram_probs. After I have calculated the probabilities according to n value for each word group, I have added them of these dictionaries. The reason I have created dictionaries for all probabilities in the dataset is to avoid calculating the probability of a specific word or word group again and again when generating sentences or calculating the probability of a given sentence. In this way, I have chosen to reduce the time the program takes in this trade-off between space complexity and time complexity. Finally, I have again used a list structure for my generated sentences to calculate their probabilities, smoothed probabilities and perplexity values.

By using the data structures I have created, many information can be found about the language model. If the list_of_sentences is called, the structure of sentences like them containing beginning and end tokens can be seen. If the n-gram dictionaries are called, common and rare words or word groups can be found since all the dictionaries are sorted. Finally, the probabilities of the words or word groups can be found easily since the probability dictionaries are also sorted.

2 Error Analysis of Generated Sentences

Before the error analysis of the generated sentences, it makes sense to introduce the functions I've implemented to generate sentences and their probability values. After building n-gram models using Ngram() function and calculating the probabilities of them using the calcProb() function, to generate sentences, I have implemented Next() and generate() function. The function generate(), returns the lists of generated sentences for all n-gram models using an another function gen_sentence(). According to n value, gen_sentence() function, generates new sentences of that model using the Next() function. Next() function takes a word parameter and if this parameter is None type, that means the function will return a word for the unigram model, but if it is one word or two words, that means the function will return a word for the bigram and trigram models. While choosing word, the next function makes a random choice using the probabilities of the possible next words as weights according to previous word parameter. While generating the sentences, if Next() function returns the end of sentence token or the length of the sentence reaches maximum, the generation process stops.

When the first sentences are generated, I have implemented the prob() and the sprob() functions to calculate their probabilities. While calculating them, I have used the log format to avoid numerical underflow. If there would be a test set, these probabilities could tell some things about the model I've created, but since the goal of this assignment is to evaluate the model by analyzing the errors of sentences one by one, to do a better error analysis, we should use the perplexity values of the sentences. Because, the probability values depends on

the length of the sentences and in sentence generation part, and since the some sentences I've generated ended early because of end of sentence token, it may not be useful to use only probabilities for error analysis. The perplexity is needed.

3 Calculation of Perplexity

To calculate perplexity of a sentence, firstly, the probability of it should be calculated. In the ppl() function, fisrt, I have calculated the probability of the given sentence, if it equals to zero, I've calculated the smoothed probability of that sentence. (There is a chance that prob() function returns 0, because, the generated sentence might not end with an end token since the length of the sentences reaches maximum, and to calculate probability, we need to add beginning and end of the sentence tokens. In this situation, the model might not contain the last bigram or trigram part of the sentence and this makes the prob() function returns zero. But sprob() solves this problem with smoothing.) After calculating the probability value, I have calculated the this value to the power of -1/N. Here, N is the number of tokens in the given sentence. An important note here is that if the model is unigram, N does not include beginning and end of the sentence tokens, and if the model is bigram or trigram, N includes the end of the sentence token but does not includes the beginning of the sentence token.

4 Results of Sentence Generation

This section contains the generated sentences and their probabilities, smoothed probabilities and perplexities.

Results of generated sentences for the unigram model:

- Sentence 1: tidy to the n't but spring them very cecilia but , . for got 's thing young sake the john king ground enough comfort of do
  Probability = -247.892963, Smoothed Probability = -247.917699, Perplexity= 741.521516
- Sentence 2: the front wife
  Probability = -33.012004, Smoothed Probability = -33.016514, Perplexity= 2053.688264
- Sentence 3: think taffy by . youth unusually . – and rosy ice to voice them more watch all best she with they daughter very with till know again to
  Probability = -271.818014, Smoothed Probability = -271.847482, Perplexity= 836.247517

Results of generated sentences for the bigram model:

- Sentence 1: fortunately for example by the red , mr. walters lands . '
  Probability = -80.444866, Smoothed Probability = -108.495441, Perplexity= 72.911103
- Sentence 2: no harm , and beat fast as much happier for him with the flag waving their questions , " continued , it had glided behind miss sara and if the young lady servants '
  Probability = -215.169640, Smoothed Probability = -270.737083, Perplexity= 70.899523
- Sentence 3: as for she entered a well pleased her again hid his life . "
  Probability = -92.353463, Smoothed Probability = -105.428890, Perplexity= 71.352770

Results of generated sentences for the trigram model:

- Sentence 1: mamma said , " we 'll remember that the workmanship was much wasted ; but the barn , and a longer one who is jonas ?
  Probability = -114.375298, Smoothed Probability = -266.525936, Perplexity= 18.845160
- Sentence 2: " we might choose .
  Probability = -24.179687, Smoothed Probability = -48.363368, Perplexity= 16.335603
- Sentence 3: but , on the mountains of ice , beyond which deliverance lay .
  Probability = -47.448274, Smoothed Probability = -125.723548, Perplexity= 10.477063

The above perplexity results show that, unigram model is not good at sentence generation. Because while generating sentences, it does not use the previous words. Bigram model is much better than the unigram model, but the best one is the trigram model. The less perplexity, the better the language model.