BBM 497: Introduction to Natural Language Processing Lab Assignment - I

Furkan KARAKÖKÇEK
21328155
furkankarakokcek@gmail.com

March 21, 2019

1 Introduction

In this assignment, we are learning the basic of n-gram language models and some calculations for the language models, sentences, probablity. N-gram language models are used to probablistic determination of word,sentence,text. This time, there are some given training data for language models and according the this constructed models we can collect some guesses about given test data.
Our dataset is The Federalist Papers and were written in 1787 to promote the ratification of the United States Constitution. Essays of Alexander Hamilton and James Madison were used for training data. According the training data, we can detect authorship of given essays.

2 Tasks

2.1 Building Language Models

This task is keystone of the next tasks and efficient and low-cost collection usage very important. I preferred dictionary for task and this dictionaries store only the count of the words at given training set. There is one dictionary for each language model and there are two different author. Names of the dictionaries like this:
c_unigram_hamilton, c_bigram_hamilton, c_trigram_hamilton
c_unigram_madison, c_bigram_madison, c_trigram_madison

For training data this essays are used which are text file:
1, 6, 7, 8, 13, 15, 16, 17, 21, 22, 23, 24, 25, 26, 27, 28, 29, 10, 14, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46

First line of essays contains the author of the essay. According to first line each word counted and stored the related dictionary with sentence boundaries. This step was simplest and most important.

2.2 Automatically Generating Essays

Generating essays were very complex and hardest part of assignment for me. Because, I removed the start boundary of sentences from unigram dictionary(I thought this idea 3 days). After starts the generating.
For unigram generating, I generate each word undependently from previous words. Probablity of each word calculated and each probablity added cumulatively until 1. This step provided the interval the word probablities between 0 and 1. There will be two stopping criteria for generating.One of them is ending boundary which has higher probablity then all other words. Other is maximum 30 word generating.
For bigram generating , firstly generating the first word according to start boundary was necessary for my implementation.Then,I generate each word dependently from first previous word. This make the works harder. Because, in each word generation possible words were changing. Then storing and calculation in each iteration was necessary for bigram generating. The other steps were repeated same as unigram generating.

For trigram generating, firstly generating the first and second word was necessary for my implementation. Rest of the steps were same as bigram generating.
After all generating words for language models, probablity of each sentences were calculated and printed the console. I preferred logarithmic sum method in this step and base of logarithm was 2.
Finally, I recognized unigram sentences were very short. I guess it was because of ending boundary probablity very high and the sentences of the unigram generating were very meaningless. Bigram and trigram sentences generally consist of 30 words. This sentences were meaningful for me because words connected the previous words. When we talk about the probablities, I can say P(unigram sentence) > P(bigram sentence) > P(trigram sentene).
While calculating the probablity of sentence, I used this formulas;

Unigram:

$$P(w_1 w_2 w_3 ... w_n) = \prod_{i}^{n} P(w_i) \tag{1}$$

Bigram:

$$P(w_1 w_2 w_3 ... w_n) = \prod_{i}^{n} P(w_i | w_{i-1}) \tag{2}$$

Trigram:

$$P(w_1 w_2 w_3 ... w_n) = \prod_{i}^{n} P(w_i | w_{i-1} w_{i-2}) \tag{3}$$

2.3 Classification and Evaluation

This task there were some unknown essays then classification was handled by using probablity of the essay. Actually, at this step probablity never calculated because logarithmic summation was used this step. After reading of essay contents, I found floating numbers such as -5541.0511786515165, -6015.156165789984, -4512.157887800158. When I try the $2^{-4512.157887800158}$, result was 0. Then comparing with this logarithmic summed values was enough for detection. For example, according to Hamilton training data, logarithmic summation value is -3925.151510189515 and according to Madison training data, logarithmic summation values is -3845.1500481109487 then comparison between this values was made and author of given test essay is Madison.This calculations and detections were made by trigram and bigram language model separately. Finally perplexity was calculated for each given test essays using again bigram and trigram language model separately.
While calculating perplexity, logarithmic summation values divided by the N which was the number of logarithmic sum operation(number of multplying probablities). But there was still huge perplexities such as 859723.3175325954, 58457679.200873815 , 44106005.40762186. This results may not be correct but helped to understand what is perplexity and important. According to perplexities of each essay bigram perplexity lower than the trigram perplexity, I understood bigram probablity of the given test essay was higher than the trigram probablity. My last inferences, perplexity grows up related with given test essay text size and unseen words.
For unseen words of the test essays, add-one smoothing was used.

3 Conclusions

My authorship detection mechanism works perfectly and finds the correct author for given test essay even if includes unseen words. My results coherent for two model bigram and trigram.Their results are enough for this task. %100 of the test essays are were found correctly. But, I guess there are time problems for task-3. Because unseen word at the given test essay really enforce the my implementation and there are many loop that search tries to find the exact key. I used the functions as possible for elegancy but there were two different authors. That makes the works harder. You can compile the my software basically :" py main.py "

4 References

1-https://nlp.stanford.edu/~wcmac/papers/20050421-smoothing-tutorial.pdf
2-https://books.google.com.tr/books?id=G_mrBwAAQBAJ&pg=PA131&lpg=PA131&
dq=add-one+(Laplace)+smoothing+python&source=bl&ots=F4yEEZeB1D&sig=
ACfU3U3KGsPEoSqiEiMke0UQqiGbH-6JLA&hl=tr&sa=X&ved=2ahUKEwiJ98uuvIzhAhWOlIsKHbV0BB4Q6AEwCXoECAgQAQ#
v=onepage&q=add-one%20(Laplace)%20smoothing%20python&f=false
3-http://www.albertauyeung.com/post/generating-ngrams-python/
4-https://towardsdatascience.com/perplexity-intuition-and-derivation-105dd481c8f3