**Francisco José Alguacil Jiménez**

**CREATE A MODEL OF NGRAM AND EVALUATE THE ACCURACY OF DIFFERENT CORPORA**

This work has consisted of developing a model of N grams using the numpy and NLTK libraries to calculate the score and perplexity of a given corpus to later comparing it with other corpora. These are the steps I’ve followed:

1. **Select corpus and preprocess.**

At the beginning I selected the wikipedia corpus to do the preprocessing by myself but it gave me some errors and I decided to use NLTK to take advantage of the corpora already in the library since it’s easier to program and more efficient. The topic I wanted to use was “news” and I searched different corpora which had news. My decision was to choose between Brown Corpus, CNN, Reuters, Wordnet and ABC. Al least I have used Brown, Reuters and Abc

import nltk

import numpy as np

nltk.download('punkt')

nltk.download('abc')

from nltk.corpus import abc

corpus = nltk.corpus.abc

sentences = abc.sents()

tokenized\_sentences = [[w.lower() for w in sent] for sent in sentences]

1. **Define the class and functions**

One of my objectif in this work is to divide it into two main parts. As the goal of the project is doing experiments, I thought it would be better if I put the robust code as class and function and there, I define N-gram model, probability, perplexity and later we just have to create a variable and assign a value to a n-gram, smoothing or discount. The code is so long, so I will not put here,but the piece of code we modify to do experiments are this:

n = 5

smoothing = 0.01

discount = 0.05

model = NgramModel(n, smoothing)

model.train(tokenized\_sentences)

generated\_text = model.generate(20) #we can introduce the number of token we want

generated\_text\_list = list(generated\_text)

print('Generated text:', ' '.join(generated\_text\_list))

and then train and generate.Thanks to having previously defined the functions and created the n-gram model, we can work easily

After doing the n-gram model, the class and preprocess, I did interpolation and back-off strategies using Laplace and absolute discounting. Interpolation happens the same, I defined all previously but if we want to change the value of lambdas at the end of the code we can.

#Interpolation

vocab = model.vocab

uni\_counts = model.context\_counts

bi\_counts = model.ngram\_counts

tri\_counts = {}

lambdas = [0.5, 0.3, 0.2]

result = model.sentence\_interpolated\_logP(test\_sentence, lambdas)

print('Interpolated log probability of test sentence "', ' '.join(test\_sentence), '":', result)

1. **Problematic**

The construction of the code has been basically trial and error. First I had to find out how the class was built, then assign the objects to the constructor. I tried to use several tools from NLTK to do the work more efficiently but they weren’t recognized by the constructor. Another problem was building the training. I started doing more complicated things and I used the tuples to do it easily, although it gave me this problem“'list' object has no attribute 'lookup'” and then I solved it. Another problem was programming the calculations because it required having good documentation and it is informed as we have the case of the probability function or logP score funcion. The rest of the problems I had were related to got an unexpected keyword argument, divisions by zero in interpolation, having called a function without having previously defined it well, errors in which functions have I to measure and test to print results, errors in concatenation, and use other variables instead of the one that actually is. Lastly, the biggest problem I've had has been adapting new functions like back-off strategies to the code that I already had and that worked. This made me have to restructure the entire code and almost start over.

1. **Experiments**

Generally speaking, the average "good" values for score of text generated and perplexity are high scores of text generated and low perplexity, indicating that the model is generating high-quality text and is coherent.

* **Score of generated text**
  + Brown Corpus: the best result is bigram model using absolute discounting with -41.55
  + Reuter corpus: the best result is trigram model using absolute discounting with -5.51
  + Abc corpus: the best result is trigram model using absolute discounting with -0.23
* **Perplexity of generated text**
  + Brown Corpus: the best result is bigram model using absolute discounting with 4.23
  + Reuter corpus: the best result is 5-gram model using absolute discounting with 12.48
  + Abc corpus: the best result is 5-gram model using absolute discounting with 0.78
* **Score of text sentence: “the news article reported on the recent economic trends'’**
  + Brown Corpus: The best result is bigram model using absolute discounting with -20.25
  + Reuters Corpus: the best result is bigram model using absolute discounting with -7.44
  + Acb Corpus: the best result is bigram model using absolute discounting with -9.54
* **Perplexity of test sentence:** 
  + Brown Corpus: the best result is bigram model using absolute discounting with 4.75
  + Reuters Corpus: the best result is a 4-gram model using absolute discounting with 19.12
  + Abc Corpus: the best result is a trigram model using absolute discounting with 30.54

In general, the model more suitable is bigram and the best method is absolute discounting. The model is capable of generating readable text from the bigram. Even so, according to my verifications, the model is not the best because it would probably need more training or perhaps a larger corpus, that is why in many ngrams models the perplexity is so high and the score is so low. I have also found that sometimes the generated text is better than what comes out in the perplexity and the score. In conclusion, I think that I could get more out of the model and that with a little more time and training better results could be achieved.