Text

Description automatically generated with medium confidence

Statistical Natural Language Processing

ICS2203

Programmed using Python

By Mark Dingli

20703H

**Table of contents**

Statement of Completion …………………………………………………………………………………… 3

Introduction …………………..…………………………………………………………………………………… 4

Design ……….…………………..…………………………………………………………………………………… 5

Observations .………………..…………………………………………………………………………………. 35

Conclusion ..…………………..…………………………………………………………………………………. 36

**Introduction**

In this documentation, I will be presenting three distinct N-gram models created using the (Baby) British National Corpus as the basis. The models include unigram, bigram, and trigram, implemented using a vanilla language model, a Laplace smoothing language model, and a language model with unknown word handling. Additionally, the documentation will provide a detailed explanation of the evaluation metrics used to compare the models' performances, including perplexity and accuracy scores. Furthermore, the advantages, testing and limitations of each model will be discussed, along with suggestions for possible improvements or modifications. The primary aim of this documentation is to provide an overview of the implementation process for each model, as well as to test and compare their respective performances on the given corpus.

**Imports**

Various Python libraries were imported and utilized. The Natural Language Toolkit (nltk) library was imported for text processing, including tokenization, lemmatization. The RegexpTokenizer class from the nltk.tokenize module was used to tokenize the text. The WordNetLemmatizer class from the nltk.stem module was used to perform lemmatization. The sklearn.model\_selection library was used to split the data into training and testing sets. Additionally, psutil library was imported to monitor the system's memory usage during the execution of the program.

**Extracting and Pre-processing the Selected Corpus**

**Description**

The corpus used in this project is the (Baby) British National Corpus, which is a collection of written and spoken texts from a variety of sources.

Initially, a single XML file from the corpus was used to develop and test the pre-processing steps. Once the pre-processing steps were finalized, the entire corpus was used for N-gram modelling.

Before the corpus was used for N-gram modelling, it underwent several pre-processing steps to ensure that it was ready for analysis. Firstly, all XML markup was removed from the corpus using the ElementTree module. Secondly, all non-alphabetic characters, such as numbers and punctuation, were removed using regular expressions. Lastly, the corpus was tokenized using the RegexpTokenizer class from the nltk.tokenize module. Additionally, the corpus text was converted to lowercase and lemmatized using the WordNetLemmatizer class from the nltk.stem module, which maps words to their base form.

It should be noted that the entire corpus was used for this project, which consisted of 4,003,540 words in total. The size of the corpus allows for a comprehensive analysis of the language patterns within it, which is essential for the creation of effective N-gram models.

**Output**

The output of this code is a list of tokens, which are the individual words or phrases that make up the text data. These tokens have been pre-processed to remove URLs, convert text to lowercase, expand contractions, remove punctuation and numbers, and lemmatize the remaining tokens. The list of tokens is printed to the console for the first 100 tokens. The total size of the extracted corpus (the number of tokens) is also outputted.

Example output of list of tokens:

A screenshot of a computer

Description automatically generated with low confidence

Figure 1 - Output showing the list of the first 100 tokens

Example output of corpus size:



Figure 2 - Output showing the total size of the extracted corpus

**Testing**

It is important to note that the pre-processing steps used in this code were selected after testing and evaluating different techniques for tokenizing and cleaning the corpus. These steps were chosen as they produced the most effective results for N-gram modelling.

The code performs several checks to ensure that the specified directory exists, contains XML files, and that each XML file can be parsed before tokenizing its text. If any errors occur, the code prints an error message to the console. Additionally, an error message is printed if the regular expression used to tokenize the text is invalid.

**Limitations**

While this code provides an effective way to pre-process the corpus, it may be slow for very large corpora, as it needs to traverse all files in the specified directory and subdirectories. Additionally, the pre-processing steps used in this code may not be suitable for all types of text data, and may need to be modified depending on the nature of the corpus.

**Splitting the corpus**

After pre-processing, the corpus was split into training and testing sets using the train\_test\_split method from the sklearn.model\_selection module. The training set was used to build the N-gram models, while the testing set was used to evaluate the performance of the models.