Text Analytics: 1st Assignment

Tsirmpas Dimitris
Droutzas Vassilis

January 27, 2024

Athens University of Economics and Business
MSc in Data Science

Contents

1 Introduction

This report will briefly discuss the theoretical background, implementation details and decisions taken for the construction of bi-gram and tri-gram models.

The full code can be found at `https://github.com/dimits-exe/textanalytics`. Note that the notebook does not contain the models, which are imported from python source code files in the `src` directory.

2 Datasets

2.1 Original Dataset

For the needs of this assignment, we picked the movie reviews corpus from the NLTK data repository, as well as a hand-picked selection of files from the Gutenberg dataset.

We followed the following Data preprocessing steps:

- We converted the text to lowercase letters.
- We used tokenization in terms of both sentences and words.
- We divided the dataset in 3 sets, the training set (60%), development set (20%) and test set (20%). We used the development set in order to get the optimal alpha value which would be used to find the bigram and trigram probabilities.
- We removed some special characters, such as [ ] ? !

We used a function to get the counters of unigrams, bigrams and trigrams. Regarding the OOV words, in the training dataset, we checked for words that appear less than 10 times. These words were filtered and their value was set to 'UNK'. (OOV words).

We initialized a new corpus, called 'replaced_corpus' , where OOV words are replaced with 'UNK'. It iterates through each sentence in the original corpus ('all_corpus') and replaces words with their corresponding "UNK" value if they are OOV.

To find the vocabulary, we simply iterated the word counter and added all the words that were not OOV. To make sure we did not include duplicates, we converted the vocabulary to a set.

The same process was applied for the development and test sets, except that now we kept the same vocabulary. We updated the sentences with the 'UNK' value when necessary. Finally, we calculated the 20 most frequent words of unigrams, bigrams and trigrams and the vocabulary length, which can be found in the notebook.

2.2 Corrupted Dataset

In order to test the spell checking models, a new dataset needed to be created. We decided to use a manually corrupted version of the combined dataset mentioned above.

Thus, we created a function `get_similar_char()` to define replacements from original characters. For example, a would be replaced by e, c would be replaced by s etc. This function returns a randomly chosen character from those defined.

The function was subsequently used by `corrupt_sentence()`, which takes as input a sentence and returns a new corrupted one with a probability for every character of 0.1 (user-defined parameter).

3 Language Modeling

To find the cross-entropy and perplexity, we used the models defined in the .py files with the simple formulas of cross entropy and perplexity in the corresponding functions.

What we needed next was to find the optimal alpha for the probability formulas, as we stated earlier. In ngram_model_alpha_search() we initialize a numpy array to store the entropy values. Iterating the alpha values, we calculate the cross entropy for each alpha. Finally, we keep the index with the best alpha (the one with the smallest cross entropy value). We searched for 1000 alpha values taken from an exponential sequence in the range of $[10^{-10}, 1]$.

4 Spell Checking

In order to obtain the WER and CER scores of a sentence, we imported the `jiwer` package, from which we used the `wer()` and `cer()` functions to calculate the corresponding scores. Then, we just took the average of these scores.