Report
Ivan Konstantinov

1. Introduction

This report aims to describe the decisions made in implementing plagiarism detection, the techniques used and the content duplications observed. The report will start with a brief overview of the program.

2. Overview

In order to properly run the program, the dataset must reside in the same directory. The directory for the real data must be called *0787128*, while the directory for the training data must be called *train*. The program can be started using the *start()* function, which allows several arguments to be passed: *_corpus* is the data to be used (either *'0787128'* or *'train'*); *algorithm* is the task to be performed (*'finn'* for Finn plateau algorithm, *'near'* for finding near duplicates, *'exact'* for finding exact duplicates, *'all'* for running all tasks); *outfile* is the name of the file to store the results (used for debugging). The default values are *'0787128'*, *'all'* and *'duplicates.txt'*.

3. Libraries

The following libraries were used: *hashlib*, *re*, *subprocess* (for reading the contents in the data directory), *sys*, *operator*, *Lab2Support* (for both reading data from file and writing data to file). The hamming distance algorithm was copied from *Wikipedia* (mainly the idea of using the *zip* function)

4. Task implementation

1. Text preprocessing

Some degree of preprocessing was used to ensure good performance of the plagiarism detector. It was observed that exact duplicates differ only by the name of the person providing the transcript (as in *"This is a transcript of a European Parliament speech by Sarah Corless"*). Since this sentence can be described as a *meta-sentence*, i.e. one providing information **about** the document, it is removed by the detector before any further processing takes place, using: **re.sub('This(\s)+is+[^\n]*transcript[^\n]*\n','',txt)**.

Additionally, punctuation is removed from the document using the regular expression **re.sub('[\.\'\^\,\:\;\"\<\>\\\/\{\}\[\]]','',str.lower(txt))**, which also converts any capital letters to lower letters (not that *'_'* or *'-'* are not removed from the text, since they may carry special meaning, such as in *'sub-process'*). As a final measure, stop words are removed from the document to ensure that they do not affect the accuracy of the algorithms (the used list of stop words does not include **all** such words).

2. Exact duplicates

The following procedure is used in order to find exact duplicates in the dataset. Firstly, each document is preprocessed using the methods described above. Then, the document is hashed using *hashlib.md5* hash function. The hash value is used to index a dictionary variable which stores all documents by their hash values. If a collision is detected, the colliding documents are *exact* duplicates and are returned in a list.

3. Near duplicates

A different approach is used to identify near duplicates. After preprocessing, a *simhash* is calculated for the document, as described in the lecture slides: each word is extracted and hashed using *md5* function, then added to the resulting document vector. In this case, each word is hashed using 112 bits: the reason for using 112 bits is to ensure high accuracy and because the *md5* hash values may not always be of the same length: for example, 'truthfully' will produce a 128-bit hash, while 'truthfully' will produce 124-bit hash value.

After computing the document vector, it is split into 28 4-bit groups. Each group is indexed into a separate dictionary and if a collision is detected, the documents in the dictionary entry are compared for similarities. The choice of 28 groups of 4 bits insures that near duplicates which produce slightly different simhashes can still be identified, as in the case of *6543219.txt* and *0270416.txt* from the training dataset.

Two algorithms were used for comparing two documents: Hamming distance and a custom Word Occurrence algorithm. While the Hamming distance algorithm computes the number of differing same-position bits in the two documents, the custom algorithm computes the similarity between the documents by finding their intersection in terms of words occurring in both documents. While the Hamming Distance approach proved to be highly accurate for nearly identical documents, the custom algorithm performed better for near duplicates with slightly different simhash values.