1/13/2020

Arapaut Sivaprasad, Ph.D.

WebGenie Software Pty Ltd.

[avs2904@webgenie.com](mailto:avs2904@webgenie.com); <https://www.linkedin.com/in/dr-siva/>

m: 0411 132 646

**STLP Project**

**- Mid-Term Analysis** **-**

# INTRODUCTION

Commercial in Confidence

# DISCLAIMER

This document is an analysis of the current situation at the half-way mark of the originally planned project costing. The estimates of CPU execution times and speeding up are approximate. Projected speedups are theoretical that are not tested yet. It is possible that we have hit the maximum speedup already. There may be inadvertent factual errors in this document. The logics and methods described in this doc are not final and may be changed significantly.

# Mid-Term Analysis

# TL;DR

* Original objective of **Cosine Similarity**: Find optimum score and high accuracy.
* - Determined that the **optimum** CS score is **0.94**
* - Achieved an **accuracy** in excess of **98%**
* Tables created by pre-processing documents with NLP are used instead of full docs.
* A modified algorithm was developed from the one for comparing full documents.
* **Semi-automatic grouping** of documents is now possible. Can make it fully automatic.
* Optimised the code and process to bring down execution times by more than **25-fold**.
* - Another **80x** improvement over it may be required to be practical.
* 3,000 docs have been analysed and grouped. They form the basis for further comparisons.
* It may require **50-60 hours** of CPU time to complete all docs in the CSV.
* - This can only be run overnight and un-attended.
* Require cloud or remote Linux servers.
* Current execution time is **20 milliseconds[[1]](#footnote-1)** per pair. It must be reduced to microseconds.
* The **Vision** is to have a software that virtually instantly classifies a new document.

## Introduction

Based on the [High Level Design Document](STLP_High_Level_Design.docx) the project was started on **16 Dec 2019**. Some prior work on the algorithm, using web pages randomly selected, had been conducted. The time spent on it was neither included in the project nor billed. It was a preliminary assessment of the program logic to make sure that the project was do-able.

Upon the commencement of the project, on or around 16th December 2019, Gavin sent the access details to **MS Azure Storage Explorer**. One of the tables, ‘**stlprecordassociationkeyphrases’**, contained the filenames and NLP-extracted keywords/phrases. This table was downloaded as a CSV file for analysis. Though it did not contain the entire content of the documents, it was correctly assumed that the **keywords/phrases** could be concatenated into a sufficiently long **text for cosine similarity** (“**CS**”).

## Details

The structure of the document is as follows. The required columns are highlighted in blue.

* **KeyPhraseLocation** : body
* **KeyPhraseLocation@type** : Edm.String
* **PartitionKey** : Derived from filename (see below)
  + https:||qsaqutpoc.sharepoint.com|shared documents|11599293\_meeting report - rti training 200109.doc
* **RowKey** : Keywords and phrases. Multiple rows
  + agency
  + annette fraser location
  + big money
  + brief tony waters
  + business value
* **SourceFileName** : Filename without path
  + 11599293\_Meeting Report - RTI Training 200109.DOC
* **SourceFileName@type** : Edm.String
* **SplitPageNumber** : Unknown and unused. Possibly the page number where the keyphrases were taken.
* **SplitPageNumber@type** : Edm.Int32
* **Timestamp** : 2019-10-26T06:08:13.042Z

## Processing

Unlike the processing of web pages, the data from the CSV file does not have the full page content. The KeyPhrases are not always extensive either. This could conceivably affect the CS score, but it remained to be seen.

There are **19,267 lines in the CSV** file. In the pre-project trials **30 URLs** were compared to each other in all possible pairs to determine the CS scores and manually examined for accuracy. Positives are those pairs that were on the same topic and negatives were different topics. A “**False Positive (FP)**” is a pair that scored above threshold even though the topics are different, and a “**False Negative (FN)**” is a pair that had the same topic but scored below threshold.

**Error rate** was calculated as a **percentage of FPs plus FNs** out of total pairs. Then, the **accuracy** is the **error percentage deducted from 100**.

Based on the best accuracy a CS score of **0.94** was chosen as the **threshold**.

The number of possible pairs is determined by the formula below.

**n\_pairs = ((n\_docs \* (n\_docs+1)) / 2 ) - n\_docs**

or

**n\_pairs = ((n\_docs \*\* 2)/2) - (n\_docs/2)**

Thus, **19,267** lines would require **185,599,011** (186 million) pair-wise comparisons. This is far too much for the algorithm developed for the URLs. Hence, optimisations to speedup the execution were required. These are described later in this document.

## Pair-wise comparison

The CSV file is read fully, or line by line, into the RAM and the rows are split for each column. By looking at **column 4** (“**SourceFileName”**) the **keyphrases** in **column 3** were concatenated for each file, using space as separator between each row. Since the filenames often had relevant key words in them, the **filenames were appended** to the begininng of the text formed from the keyphrases. This proved to assist in the scoring (see later).

The results were added to an ‘**OrderedDict’**[[2]](#footnote-2) object using the **SourceFileName** as the **key** and the concatenated **keyphrases** as the **value** for each file.

The comparison, using **parallelised** **code** to work across **multiple cores**, was then made using the same code for comparing the page contents from URLs. The CS scores so calculated are kept in a **shared memory** across multiple processes and, upon termination of all processes, re-written into a **TSV** file for manual analysis.

The manual analysis was done by reading the TSV file into Excel, highlighting the CS scores above threshold (tried **0.90** to **0.95** and decided on **0.93** as final) and then manually comparing the file names and the keyphrases between the members of each pair of documents. This analysis is **subjective**, but in most cases the filenames were indicative of whether the docs were similar or not.

Based on these comparisons the scores were marked as **True**, **False Positives** and **False Negatives**. After analysing 200 lines from the CSV it became clear that a cut-off **CS** score of **0.94** was the optimum that gave more than **98% accuracy** based on false positives and false negatives.

It was noted that **pair-wise** comparison **cannot be scaled up** and hence this method was of use only with small numbers of documents for arriving at initial **groupings**. These groups will form the basis for further comparisons.

## Jaccard Similarity

With a view to supplement the CS score, the Jaccard Similarity (JS) was calculated between the documents in each pair. Initially it seemed that all those above a CS score of **0.94** had a JS score above **0.2**. Later, however, it was found to be not true and hence the JS score was removed from analysis.

## Optimisations

Initially it took an average of **0.5 seconds** per pair to process. This was partially due to race conditions caused by multiple processes competing for I/O. Applying locks removed such race conditions and the time was reduced to **0.3 seconds** per pair. It was still too high to compare in all possible pairs. For example, **500** docs would take more than **10 hours** to complete. It was therefore necessary to optimise further.

Another way to reduce the times was to **omit a doc** from further comparisons once it has scored **above threshold** with another doc. For example, doc1 may give lower than threshold (0.94) for up to the doc10. Then, if doc11 scores above threshold, the doc11 is added as a member of the doc1 group and then removed from further comparisons. By applying this logic, the times were reduced to **0.1** seconds. Still it is far too long for processing millions of pairs.

It was noted that this method would **miss** several documents from the initial groupings. It can be handled later by comparisons of **groups against each other**.

## Grouping

Both to reduce the execution times and to start building categories of documents, those pairs which gave above threshold were put into **groups**. Thus, if doc1 has compared with docs 2, 5 and 7 then doc1 becomes the ‘**reference doc**’ and the others become its ‘**members’**.

Such groupings provide a way to reduce the number of comparisons. After building a set of 70 reference docs, with multiple members under each reference, a different set of docs were compared to the initial reference docs. This reduced the number of comparisons to a maximum of a multiple of the number of references against the number of new docs. For example, analysing **500** new docs against **70** references would take a maximum of **35,000** (often less if a match is found early) instead of the **124,750** pairs it would otherwise take.

## Ungrouped docs

The above logic classified about **40%** of 3000 docs into **groups**, leaving **60% not** belonging to any group. It is expected that when all 19,267 docs are finished there will be no ungrouped docs. This is yet to be seen.

As the number of reference docs increase, so does the time to analyse the new docs with the reference. There are currently **335** separate groups and a total of **1,693** members out of 3,000 docs analysed (**67.6%**). So, it appears that the number of ungrouped docs will come down.

## Secondary Grouping

The 335 groups represent an unacceptably large number of categories. We must reduce this to as few as possible. One possible way to do it is by manually assigning a code to the reference docs. Even if the reference docs do not match each other, with manual examination it may be possible to say that two or more docs belong to the same “**Higher Group**”.

The above strategy will still have difficulties. A new doc may have to be compared with more than one reference doc before saying that it belongs to a higher group.

Another method to be tried is to create a “**Super Group**” by concatenating the key-phrases of the group and its members together. These can then be compared against each other. It will, hopefully, **pick up matches** that were missed previously and help to reduce the number of groups.

# Where to go from here

The immediate objective is to analyse all 19,267 docs and assign them to **lower** and **higher** groups. It will take approximately **10 hours to process 5,000 new docs** against the currently available 335 reference docs. Once this comparison is made, it will increase the reference docs and therefore the next 5,000 new docs will take longer. To complete the entire set, we could be looking at **50-60** hours of CPU time plus several hours of manual processing.

The processing is currently done on the same as the development machine. When it is running, it uses up 100% of CPU cycles and no other work is possible. We must separate the development and execution to separate servers, ideally remote Linux servers.

## The Goal

The goal is to have **a system ready for comparing new docs as they come in** and assign them to a lower, higher group and a top-level group or “**Record Authority**”. Both the accuracy and execution times are important considerations, as there could be thousands or more docs coming in simultaneously. We must experiment and devise better ways to optimise and speed up the comparisons.

Using an “**inclusive/exclusive**” keyword may help to place a doc in a category without calculating the CS score, but it may involve too much **manual work** to add these to every category. Neither will it be accurate, as a doc may miss it but is still within that group.

Other methods such as looking at the header, logo, etc. may also help. These are not depended on CS scoring and are **not part** of the current project.

## Speeding up execution

Reducing the time to compare two docs by changing the methods and/or caching intermediate values must be tried. Though the present code is **25-fold faster** than the original that was developed using web pages, it should be speeded up by another **10** to **80**-fold (i.e. **250** to **2,000**-fold over the original).

## The Vision

Though I have not seen the original documents, a general idea about their content and format is available from the CSV file. When a new document, or hundreds of documents, arrive they should be compared with a finite number of reference docs to classify. **This must happen in microseconds.** The current average of **20 milliseconds** will be far too slow when it comes to hundreds, let alone thousands, of new documents.

There is a large **I/O** component at present, which is required to monitor during development. Perhaps the execution times will improve dramatically when these I/O are eliminated.

The intermediate step of **NLP-processing** to get the keywords/phrases could become a bottleneck. Must revisit my original code to work out how much overhead is involved in vectorising the raw document versus keyphrases.

## Still to do

The software is still on the development platform and is run interactively. It must be installed on remote servers and accessed via **APIs** or **Web**. Such installation is currently impeded by the inability to install all the required Python modules on my Linux servers. It must be investigated further[[3]](#footnote-3).

\_\_\_\_\_\_\_\_\_\_\_END OF DOCUMENT\_\_\_\_\_\_\_\_\_\_\_

1. This has now been reduced to 6 milliseconds [↑](#footnote-ref-1)
2. An ordered dictionary will retain the order in which items are added. It was chosen instead of normal dictionary for the ease of comparisons that followed. [↑](#footnote-ref-2)
3. A Linux server has been setup to run the software from command-line. [↑](#footnote-ref-3)