12/2/2019

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**

**- High Level Design** **-**

# INTRODUCTION

Commercial in Confidence

# DISCLAIMER

This document is a high-level design of the project and is based on the current understanding of the requirements and methodology. It primarily deals with analysing the content of web pages for similarity, which is applicable for locally stored text files as well. It may not be useful in classifying documents of dissimilar contents into the same category. Changes in the algorithm may become necessary as we progress.

# High Level Design Document

## Introduction

Classifying documents into categories based on their contents is the subject of this paper. The first task is to devise an algorithm to read the contents of two docs and make a similarity score. Based on the similarity score one can say how well these are related. For example, two separate web sites dealing with the same topic (e.g. ‘Machine Learning’) may give a similarity score of 0.95 which could be taken as indication that they belong to the same category. We may set a cut-off score of, say, 0.9 to distinguish between similar and dissimilar docs.

However, there could be situations where two docs may give a score close to either side of 0.9, in which case other criteria must be used to diverge the scores.

Separate categories could be established with one or more documents having contents that are typical of the category. New docs are then compared with the reference doc(s) to determine whether they belong to the category.

## Objectives and Expectations

The ultimate objective is to classify thousands of documents, stored in various formats, into their appropriate categories and assign to a “Records Authority”. This may include docs of similar contents, similar source but different contents, similar properties but different sources and contents, etc. While it is the goal to identify and categorise all these types of docs, the focus of this current project is to classify only the docs of similar contents.

The expectation is that a confidence level of >90%, even 95-98%, can be achieved. The industry average is around 80% (?) in this area.

## Current methodology

“*Castlepoint is a single solution to manage all the records in your organisation. It registers every record in every business system and uses Artificial Intelligence to classify it against a Records Authority and apply a sentence. It acts as a single pane of glass to find, relate, manage and audit every record in your network, no matter what system it is stored in – and it does this without any impact on your existing systems or your user base*.”[[Ref](mailto:https://www.castlepoint.systems/about/)]

“*Castlepoint extracts key phrases from the content of every piece of information using Natural Language Processing, and uses this data to determine its subject, and its matching Records Authority classes and corresponding sentence*.”[[Ref](mailto:https://www.castlepoint.systems/usability/)]

The existing technology is based on keywords/phrases. Documents in various formats (Word, PDF, HTML, scanned docs, etc.) can be analysed. Details of the algorithm are unknown but appears to be using word density and key phrases.

Other criteria may also be in use. For example, the logo on a page could indicate which organisation/department it belongs to. The metadata (e.g. in emails) may also help in categorising.

## System Overview

The new system will use Natural Language Processing (**NLP**) to analyse documents in pairs. The diagram below summarises the process flow.

**Fig. 1**

Reference Doc

New Doc

NLP

Score

Above threshold

Borderline

Below threshold

Word density

Add to category

New category

It will be required to analyse hundreds or even thousands of documents to arrive at a threshold score. Machine learning during this phase may help generate categories automatically. We will do enough comparisons so that no new category is being created. This may be impossible with web pages, but an organisation may only have finite number of categories. The expectation is that the number of final categories will be low enough for manual assignment. One or more category may be assigned to the same “Record Authority”.

New documents will be compared against one or more docs in each category. If using more than one as reference doc, then the average value of score may be used (to be determined empirically). If a doc falls into more than one category, then other methods must be employed. Relative frequency of the most common words between the docs is one way to go. This must be optimised.

Other tests such as source of the doc, logo, other images, links, etc. may help as well. These are not part of the current project but will be tried if time permits.

## System Architecture

The system will be a Python program that resides on a Linux server. The main programming language will be Python but Perl, GO and C may also be used.

Jupyter notebook or Pycharm will be the testing/development platforms. It may require the installation of a frameworks such as Anaconda and Pycharm on a local PC.

A web interface and command-line execution will be provided. The web interface will use HTML/JavaScript/AJAX and will be functional but not fancy.

Comparing two docs using cosine similarity is a CPU-intensive task and hence parallelisation is essential. Python’s multiprocessing, instead of multithreading, may be more appropriate for parallel processing in this case. The latter is good for I/O bound programs but, even though there will be significant I/O waits in reading the docs, Python’s Global Interpreter Lock (GIL) will only allow the execution of one thread at a time and will negate the benefits of multithreading over multiprocessing.

## Data Design

The data will come as URLs or local text files. In this project only text and html formats will be handled. It can handle other formats if there are facilities to convert them (e.g. PDF) into text documents but it may not form part of the current project.

The system will not be using any keyword/phrase data for comparing the docs. They may be used later if the algorithm requires it.

Results and the logging of comparisons will be stored in a MySQL database or as plain text files.

## Algorithm

The initial algorithm will be exclusively based on NLP techniques. It must be refined by trying out different methods. Significant R&D may be required initially.

The core program will read the page (web or local doc) and vectorise the content. Prior to it the content will be tokenised, lemmatised and split into small chunks. These chunks will be vectorised and a ‘cosine similarity’ (CS) will be calculated. This will give a score between 0 and 1 based on how close the docs are.

For training of the model, hundreds of documents will be manually picked and compared against each other in all combinations. By checking the score against a table of expected results we can arrive at the best algorithm. The confidence level will be based on the CS scores that match expectations. It could be a percentage of SC scores that match the expected results.

If the initial algorithm gives a confidence level above a threshold there won’t be a need for more tests. If not, other tests like word frequency may be applied. Confidence of 90% is the base and will aim for 95‑98%. Preliminary testing has given 92% but it must be refined and repeated many times over.

## Component Design

The main component of the system will be a Python program that uses multiprocessing for parallel execution. Since the pairwise comparisons of docs do not depend on other pairs, it will be what is called an “embarrassingly parallel” algorithm. The results from multiple processes will be combined via a special array that works across processes. The program will reside on a PC or Linux server having high speed internet access.

Development will use Pycharm, Jupyter notebook or a plain text editor. Testing and execution will initially be via commandline. Towards the end of development, a functional web interface may be created.

## Human Interface Design

Initially it will be commandline execution from a shell window or via IDEs such as Pycharm. A functional web interface will be created at the end.

## Requirements

* A laptop or desktop for development.
* A Linux server for publishing.
* Multiple Python libraries to be installed.
* An IDE like Pycharm or Anaconda.
* Very large number of documents as URLs or text files.
* A web server.
* Email communication facility.
* Facility for logging and exchange of information.

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