11/12/2019

Arapaut Sivaprasad, Ph.D.

WebGenie Software Pty Ltd.

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

**STLP Project**

Natural Language Processing to Classify Documents

# INTRODUCTION

While humans can read and understand what a document is about, for a computer it is not that simple. Natural Language Processing (NLP) addresses this problem to some extent but is still not always right.

When there are hundreds or thousands of documents to be classified based on their contents, it becomes impractical for a human being to do so. Several criteria can be used to identify a document. Similarity between two documents using NLP techniques is the subject of this paper. Enhancements to achieve a high confidence level are also discussed.

To demonstrate we will be using web pages to identify those with related contents. The methods will be equally useful to local documents in text format.

# DISCLAIMER

This document is a preliminary assessment of some techniques and is no more than a collection of thoughts. It has been written prior to seeing the sample data/documents and may prove to be not applicable to such data. The principles discussed here may be modifiable to suit any type of documents. No claim is made as to the accuracy of the results which may be improved by applying other techniques not yet thought of. The example programs are written in a combination of different languages, with Python being the core language. No guarantee is given that the code can be ported to another language, but it would not be difficult to use another language to create the main program which in turn calls the Python programs in the background.

# OUTLINE of METHOD

Comparing documents to other documents in a set using cosine similarity will give a start value (e.g. 80% similarity). This, however, is not enough to say that a document belongs to a category.

Other characteristics are required to be more confident. For example, high level of match in the top set of words in each document, similarity in the source of data, identical logo, similar images, etc. will add to the confidence level.

We will be aiming for 95% or above confidence level.

## Tests implemented so far:

1. Tokenization
2. Stop word removal
3. Lemmatisation
4. Vectorisation
5. Tf-idf
6. Cosine Similarity.

## More Tests:

1. Word frequency
2. Source of data
3. Image analyses
4. Keywords
5. Metadata analysis

# PROGRAM BASICS

The Python language has a comprehensive set of modules to do NLP. Complementing it with custom methods may give us the desired results.

In the examples tried, we are using web pages as source of data. Unlike a text document, the web pages will have hidden text and formatting tags. These can sometimes assist the comparison but, in most cases, they will be just noise. We will try it first by removing the noise.

Tokenising the document into separate words is the next step. We then remove the common words like, ‘the’, ‘a’, ‘it’, etc. Also removed are numbers, punctuations, non-alpha characters and words of 3 letters or less.

Checking the Part of Speech (POS) will also help to eliminate a few other tokens. This must be tested empirically. For example, while removing the proper names of persons may work in some cases, there may be situations where they are important in the classification.

Instead of taking the whole words, which can take several forms such as tenses and other modifications, we will take the stems or lemma of the words. While there is a small chance of over stemming and under stemming, it is usually enough that related words map to the same stem, even if this stem is not in itself a valid root.

Vectorising the words will help compare them statistically. It forms the basis for the Tf-idf analysis.

Finally, comparing the text for cosine similarity will give a value from 0 to 1. It is probably necessary to break the document into small chunks before applying the similarity tests. Then, combining the values into one final value may be the way to go. Comparing two large documents may be biased. The appropriate size of text to compare must be determined empirically.

The above tests are probably the maximum extent to which Python’s libraries can go. We must go beyond it to be sure about the classification.

One test being tried is to use word frequency in the document (cf. early search engines). From the tokenised and lemmatised word list we can build a word frequency array and then take several words from the top and compare how the two docs match.

Other tests may include comparing the meta data, images and logo.

# DEMO AND PROTOTYPE

A Python program was created to demonstrate the above functionality, but currently it works only on my personal laptop. I am trying to put it up on a public webserver, but it is running into some issues.

Testing with a few URLs showed the following similarity scores. In all cases the system has correctly rated.

## URLs

1. https://en.wikipedia.org/wiki/Machine\_learning
2. https://www.sas.com/en\_au/insights/analytics/machine-learning.html
3. https://en.wikipedia.org/wiki/SpaceX
4. https://en.wikipedia.org/wiki/London
5. https://en.wikipedia.org/wiki/England
6. https://www.webgenie.com/details.html
7. https://en.wikipedia.org/wiki/Titanic\_(1997\_film)
8. https://en.wikipedia.org/wiki/Titanic\_(1953\_film)
9. https://simple.wikipedia.org/wiki/Titanic\_(1997\_movie)
10. https://en.wikipedia.org/wiki/Star\_Wars
11. https://en.wikipedia.org/wiki/List\_of\_Star\_Wars\_films

## Results

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ## | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 1 | 1.00 |  |  |  |  |  |  |  |  |  |  |
| 2 | 0.96 | 1.00 |  |  |  |  |  |  |  |  |  |
| 3 | 0.86 | 0.88 | 1.00 |  |  |  |  |  |  |  |  |
| 4 | 0.85 | 0.86 | 0.87 | 1.00 |  |  |  |  |  |  |  |
| 5 | 0.83 | 0.82 | 0.85 | 0.93 | 1.00 |  |  |  |  |  |  |
| 6 | 0.88 | 0.91 | 0.79 | 0.85 | 0.77 | 1.00 |  |  |  |  |  |
| 7 | 0.82 | 0.83 | 0.87 | 0.86 | 0.84 | 0.84 | 1.00 |  |  |  |  |
| 8 | 0.81 | 0.84 | 0.85 | 0.87 | 0.87 | 0.81 | 0.94 | 1.00 |  |  |  |
| 9 | 0.80 | 0.82 | 0.85 | 0.87 | 0.86 | 0.80 | 0.96 | 0.94 | 1.00 |  |  |
| 10 | 0.83 | 0.84 | 0.83 | 0.84 | 0.84 | 0.75 | 0.90 | 0.89 | 0.89 | 1.00 |  |
| 11 | 0.72 | 0.72 | 0.78 | 0.79 | 0.79 | 0.70 | 0.89 | 0.88 | 0.89 | 0.95 | 1.00 |

* Comparing the URL to itself always gave 1.00 as expected.
* All related URLs scored > 0.9 (green cells)
  + The closer the page contents the higher the score.
* All unrelated URLs scored < 0.9 (white cells)
* Distantly related pages scored close to 0.9 (orange cells)
  + URLs 1, 2 and 6 are about programming and, hence, scored above 0.9
  + URLs 7 to 11 are about movies, though separate ones.

Pending tests with more URLs, we can tentatively say that 0.9 is the cut-off for related pages. However, some distantly related pages are close to 0.9 and may be false positives. To eliminate or confirm these we must use other tests as mentioned above as “More Tests”. The scores could be diverged further by using more tests.

# WAY FORWARD

The objective is to add a new doc into an existing category. We can either create the categories manually and include one or more reference doc in each category. Alternatively, machine learning can be applied so that the system learns and creates the categories itself. Some manual cleansing may be useful but would not be essential.

A web interface to test the system is under construction. Currently having problems with Python installation on my server. It would be helpful if there is a server that already has the necessary Python libraries.

The production version will run in command-line or batch mode.

URLs, emails and text files will be supported initially. Converting from other formats will require third party software tools.

## Constraints

1. Sparse data in a doc can adversely affect the classification.
2. The docs must be in English and without too many spelling errors.
3. Transcribing a handwritten or PDF document via OCR software may have spelling errors.
4. Image analysis is not possible with this algorithm.
5. Contents may vary widely in same category of docs. They will not be classified as together.

# ENHANCEMENTS

## Adding word frequency score

Word frequencies are calculated for both docs and sorted in descending order. The values for each word in doc1, expressed in percentage of total words, are checked against the words in doc2. Only the top 10 words are used. The sum of the values as a percentage of the total value for these 10 words is added to the similarity score.

Thus, a similarity score of 0.95 may be enhanced to 1.65 if the word frequency score is 0.7. For unrelated docs the word frequency score will be lower or 0. Their similarity score will therefore not be enhanced as much or not at all. This way, the scores are diverged.

A score of 1.25 or higher indicates similar docs (green cells) in the table below. False positives in previous method are gone.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| # | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 1 | 2.00 | 1.81 | 0.86 | 0.83 | 0.81 | 0.89 | 0.80 | 0.80 | 0.80 | 0.79 | 0.71 |
| 2 |  | 2.00 | 0.87 | 0.84 | 0.80 | 0.91 | 0.81 | 0.82 | 0.82 | 0.79 | 0.72 |
| 3 |  |  | 2.00 | 1.00 | 0.99 | 0.78 | 1.06 | 0.85 | 1.01 | 1.06 | 0.93 |
| 4 |  |  |  | 2.00 | 1.53 | 0.78 | 1.02 | 0.86 | 1.03 | 1.01 | 1.00 |
| 5 |  |  |  |  | 2.00 | 0.76 | 1.00 | 0.85 | 1.01 | 0.98 | 0.96 |
| 6 |  |  |  |  |  | 2.00 | 0.80 | 0.72 | 0.79 | 0.75 | 0.68 |
| 7 |  |  |  |  |  |  | 2.00 | 1.40 | 1.61 | 1.20 | 1.21 |
| 8 |  |  |  |  |  |  |  | 2.00 | 1.30 | 1.04 | 1.04 |
| 9 |  |  |  |  |  |  |  |  | 2.00 | 0.94 | 0.95 |
| 10 |  |  |  |  |  |  |  |  |  | 2.00 | 1.79 |
| 11 |  |  |  |  |  |  |  |  |  |  | 2.00 |

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