Document Similarity

By Yogesh H. Kulkarni

Comparison between things, like clothes, food, products and even people, is an integral part of our everyday life. It is done by assessing similarity (or differences) between two or more things. Apart from its usual usage as an aid in selecting a thing-product, the comparisons are useful in searching things 'similar' to what you have and in classifying things based on similarity. This post describes a specific use-case of finding similarity between two documents.

Measuring Similarity

Measure of similarity can be qualitative and/or quantitative. In qualitative, the assessment is done against subjective criteria such as theme, sentiment, overall meaning, etc. In the quantitative, numerical parameters such as length of the document, number of keywords, common words, etc. are compared. The process is carried out in two steps, as mentioned below:

- Vectorization: Transform the documents into a vector of numbers. Following are some of the popular numbers(measures): TF (Term Frequency), IDF (Inverse Document Frequency) and TF*IDF.
- Distance Computation: Compute the cosine similarity between the document vector. As we know, the cosine (dot product) of the same vectors is 1, dissimilar/perpendicular ones are 0, so the dot product of two vector-documents is some value between 0 and 1, which is the measure of similarity amongst them.

Test-case used in this post is of finding similarity between two news reports [^1, ^2] of a recent bus accident (Sources mentioned in the References). Programming language 'Python' and its Natural Language Toolkit library 'nltk' [^3] are primarily used here. The similarity analysis is done in steps as mentioned below.

Documents Pre-Processing

The news reports contain many things which are not core (or irrelevant) for text analysis exercise such as finding similarity. So, they are pre-processed by converting their words into lower case and removing the 'stopwords', like 'the', 'should', etc.

```
import nltk
from nltk.corpus import stopwords

stopwords_en = stopwords.words("english")

def preprocessing(raw):
    wordlist = nltk.word_tokenize(raw)
    text = [w.lower() for w in wordlist if w not in stopwords_en]
    return text

f1 = open('buscrash_foxnews.txt', 'r', encoding="utf8")
text1 = preprocessing(f1.read())

f2 = open('buscrash_reuters.txt', 'r', encoding="utf8")
text2 = preprocessing(f2.read())
```

Vectorization

Characterize each text as a vector. Each text has some common and some uncommon words compared to each other. To account for all possibilities, a word set is formed which consists of words from both the documents. A simplest way to create the vectors is to count number of times each word from the common word set, occurs in individual document.

```
from nltk.probability import FreqDist

word_set = set(text1).union(set(text2))

freqd_text1 = FreqDist(text1)
text1_count_dict = dict.fromkeys(word_set,0)
for word in text1:
    text1_count_dict[word] = freqd_text1[word]

freqd_text2 = FreqDist(text2)
text2_count_dict = dict.fromkeys(word_set,0)
for word in text2:
    text2_count_dict[word] = freqd_text2[word]
```

FreqDist counts the number of occurrence of a word in the given text. So, in the above code snippet text1_count_dict has word-count pairs of all the words from the common word_set, along with their individual counts. Following table shows few words with their frequencies:

	westbound	whether	windows	workers	worse	would	years
text1	1	0	1	0	0	1	0
text2	1	1	0	1	1	1	1

These vectors, in a crude way, represent their respective texts and similarity can be assessed amongst them. This is the ‘Containment Ratio’ method mentioned above. TF-IDF is much better measure to represent a document.

TF is document specific. It is a way to score the importance of words (or "terms") in a document based on how frequently they appear. If a word appears frequently in a document, it's important, it gets a high score. Although it is easy to compute, it is ambiguous (‘green’ the colour and ‘green’ the person’s name is not differentiated).

```
# TF calculations
freqd_text1 = FreqDist(text1)
text1_length = len(text1)
text1_tf_dict = dict.fromkeys(word_set,0)
for word in text1:
    text1_tf_dict[word] = freqd_text1[word]/text1_length

freqd_text2 = FreqDist(text2)
text2_length = len(text2)
text2_tf_dict = dict.fromkeys(word_set,0)
for word in text2:
    text2_tf_dict[word] = freqd_text2[word]/text2_length
```

IDF is for the whole collection. It is a way to score how many times a word occurs across multiple documents. If a word appears in many documents, it's not a unique identifier, thus gets a lower score.

```
# IDF calculations
text12_idf_dict = dict.fromkeys(word_set,0)
text12_length = 2 # 2 documents
for word in text12_idf_dict.keys():
    if word in text1:
        text12_idf_dict[word] += 1
    if word in text2:
        text12_idf_dict[word] += 1

import math
for word, val in text12_idf_dict.items():
    text12_idf_dict[word] = 1 + math.log(text12_length/(float(val)))
```

TFIDF of a word = (TF of the word) * (IDF of the word)

```
# TF-IDF Calculations
text1_tfidf_dict = dict.fromkeys(word_set,0)
for word in text1:
    text1_tfidf_dict[word] = (text1_tf_dict[word])*(text12_idf_dict[word])

text2_tfidf_dict = dict.fromkeys(word_set,0)
for word in text2:
    text2_tfidf_dict[word] = (text2_tf_dict[word])*(text12_idf_dict[word])
```

Distance computation

Following are the steps to compute the cosine distance between vectors of two texts.

```
# Compute Cosine distance
v1 = list(text1_tfidf_dict.values())
v2 = list(text2_tfidf_dict.values())
similarity = 1 - nltk.cluster.cosine_distance(v1,v2)
print("Similarity Index: {:4.2f} %".format(similarity*100))
```

For the given two news items the similarity score came to about **72.62 %.**

Conclusion

TFIDF is thus a quick measure of assessing similarity, but rather a crude one. Further refinement can be brought to this analysis using topic modelling, thematic summarization of the news items, etc.

References

1. News Source 1: http://www.ndtv.com/world-news/at-least-13-killed-in-california-tour-bus-crash-report-1478120?pfrom=home-topstories
2. News Source 2: http://www.foxnews.com/us/2016/10/23/3-dead-in-california-tour-bus-semi-truck-collision.html
3. Nltk : nltk.org https://pythonprogramming.net/tokenizing-words-sentences-nltk-tutorial/
4. Computing Document Similarity with NLTK (March 2014) https://www.youtube.com/watch?v=FfLo5OHBwTo
5. Tutorial: Finding Important Words in Text Using TF-IDF http://stevenloria.com/finding-important-words-in-a-document-using-tf-idf/

Yogesh H. Kulkarni is a PhD Student doing research in Geometric Modelling. He is also intersected in Data Sciences, especially in Natural Language Processing. Please send in your comments and suggestions about this article to him at yogeshkulkarni@yahoo.com