Implementation of Text Similarity using Cosine Similarity Method in Python

Ahmad Farhan AlShammari
Department of Computer and Information Systems
College of Business Studies, PAAET
Kuwait

ABSTRACT
The goal of this research is to develop a text similarity program using cosine similarity method in Python. The steps of text similarity process are: preprocessing text, word-tokenization, creating list of words, creating bag of words, calculating word frequency, and calculating cosine similarity. The developed program was examined on two experimental texts from Wikipedia. The program successfully performed the steps of text similarity and provided the required results.

Keywords
Artificial Intelligence, Machine Learning, Text Similarity, Natural Language Processing, Word-Tokenization, Word Frequency, Cosine Similarity, Python, Programming.

1. INTRODUCTION
Now, with the huge amount of data in the Internet, it is important to find the similarity between texts. The text similarity process is applied in search engines, information systems, and digital libraries. Automatic text similarity programs are useful and help users to save time and effort especially with long documents.

Text similarity is one of the important applications of machine learning. Machine learning (ML) is a branch of Artificial Intelligence (AI) which is focused on developing algorithms to improve the performance of computer programs.

Text similarity is a common field between machine learning and Natural Language Processing (NLP). It applies both the techniques of NLP and the methods of ML to process text.

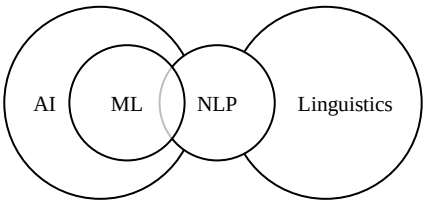

Fig 1: Field of Text Similarity

Text similarity is widely used a lot in many applications, for example: searching, ranking, clustering, classification and recommendation.

2. LIREATURE REVIEW
The early research in text similarity started in the sixties. It was originally focused on Information Retrieval (IR) (Hotho, Nürnberger, & Paass [1]). Salton, he was called the father of IR, developed the first "automatic" information retrieval system, SMART (Salton & Lesk [2], and Salton [3]).
The concept of "Vector Space Model" was introduced to represent text in the vector form (Salton, Wong, & Yang [4]). A text can be represented as a vector in the following form:

$$V = (w_1, w_2, \ldots, w_n)$$

Where: $w_1, w_2, \ldots, w_n$ are "weights" given to the words in the text.

Researchers developed different variations of weighting methods based on "word frequency" such as TF-IDF method (Salton & Buckley [5]).

The "cosine similarity" method was applied to measure the similarity between vectors (Salton & McGill [6], and Salton, Allan, & Buckley [7]).

The fundamental concepts of text similarity are explained in the following section:

Text Similarity:
Text similarity is the process of measuring the similarity between texts to determine if the texts are similar or not.

Methods of Text Similarity:
Text similarity can be measured by different methods such as: Euclidean distance, Manhattan distance, Jacquard distance and cosine similarity. Cosine similarity is the most widely used method because of its simplicity and effectiveness.

This research, will apply the cosine similarity method.

Preprocessing Text:
The raw text should be "pre-processed" first to remove the unwanted characters or words, for example: punctuation characters and stopwords.

Word Tokenization:
Word Tokenization is the process of breaking text into smaller units "tokens" (words).

List of Words:
List of words is the list of words in the text after removing stopwords.

Bag of Words:
Bag of words (BoW) is the set of words in the text without repetition.

Word Frequency:
Word frequency is the number of times a word occurs in the text divided by the number of words in the text. It is calculated

by the following formula:

$$Freq(w_i) = \frac{f(w_i)}{Nw} \qquad (1)$$

Where: $f(w_i)$ is the number of times the word (w_i) occurs in the text, and Nw is the total number of words in the text.

Cosine Similarity:

Cosine similarity is a mathematical method used to measure the similarity between texts. The concept is derived from the calculus of vectors in mathematics.

For example, consider two vectors *A* and *B* in plane, as shown in the following diagram:

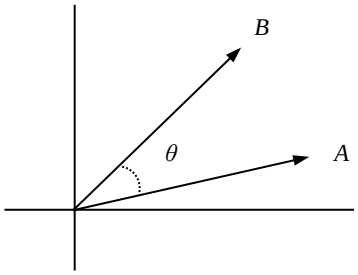

Fig 2: Representation of Vectors A and B

The dot product of the two vectors (*A.B*) is calculated by the following formula:

$$A \cdot B = \|A\| \, \|B\| \, Cos(\theta) \qquad (2)$$

Where: ||*A*|| and ||*B*|| are the norms of vectors *A* and *B* respectively, and θ is the angle between the two vectors.

Then, the cosine of the angle is calculated by the following formula:

$$Cos(\theta) = \frac{A \cdot B}{\|A\| \, \|B\|} \qquad (3)$$

In general, for any two vectors *A* and *B* in space, where:

$$A = (a_1, a_2, \ldots, a_n)$$
$$B = (b_1, b_2, \ldots, b_n)$$

Then, the cosine of the angle is calculated by the following formula:

$$Cos(\theta) = \frac{\sum(a_i \, b_i)}{\sqrt{\sum a_i^2 \sum b_i^2}} \qquad (4)$$

Where: a_i and b_i are the vector values of vectors *A* and *B* respectively.

The cosine similarity value shows the "percentage" of similarity between the two vectors. For example: if the cosine value is close to (1), then the two vectors are similar, and if the cosine value is close to (0), then the two vectors are not similar.

Python:

Python [8] is a general high-level programming language. It is simple, easy to learn, and powerful. It is the best choice for many programmers, especially in the field of machine learning.

Python provides additional libraries for processing numbers, plots, texts, and images, such as: Numpy [9], Pandas [10], Matplotlib [11], NLTK [12], and Scikit [13].

This research, will apply the standard functions of Python without any additional library.

3. RESEARCH METHODOLOGY

Text similarity is done by the following steps: preprocessing text, word-tokenization, creating list of words, creating bag of words, calculating word frequency, and calculating cosine similarity.

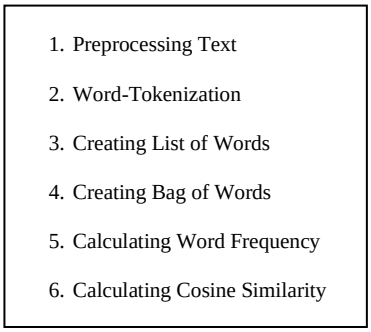

Fig 3: Steps of Text Similarity

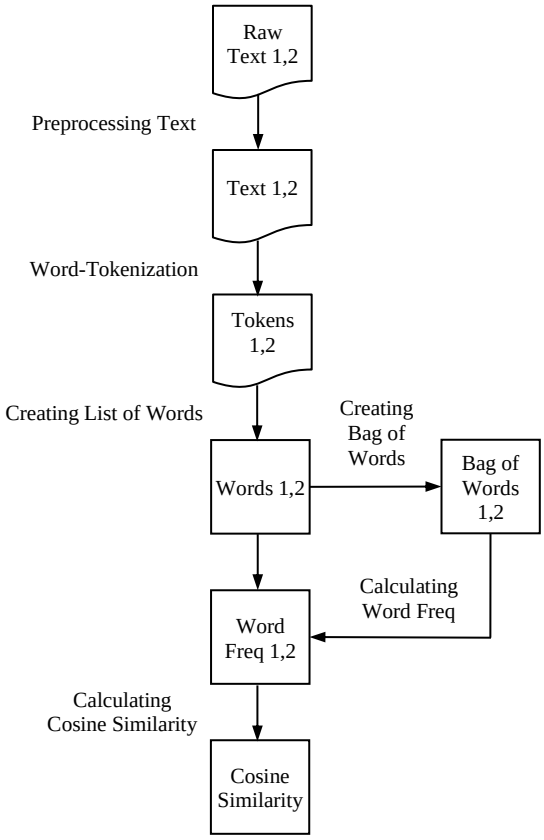

Fig 4: Flowchart of Text Similarity

The steps of text similarity are explained in the following section:

1.Preprocessing Text:

The text is preprocessed to remove the unwanted characters and

words. It is done by the following steps:

1.1 Converting Text into Lower Case:

The text is converted into lower case form. It is done by the following code:

```
text = raw_text.lower()
```

1.2 Removing Punctuation Symbols:

The punctuation symbols are removed from the text. It is done by the following code:

```
letters = "abcdefghijklmnopqrstuvwxyz"
for c in text:
    if (c not in letters):
        text = text.replace(c," ")
```

1.3 Defining Stopwords:

Stopwords are words that have no importance in the text. For example: "i", "he", "she", "it", etc. The stopwords list is defined by the following code:

```
stopwords = ["i", "we", "am", "he", "she",
             "it", "is", "are", "the",
             "they", "that", "this", ... ]
```

2. Word-Tokenization:

The text is split into tokens (words). It is done by the following code:

```
tokens = text.split()
```

3. Creating List of Words:

The list of words is created by removing the stopwords. It is done by the following code:

```
words = []
for word in tokens:
    word = word.strip()
    if (word != "" and word not in stopwords):
        words.append(word)
```

4. Creating Bag of Words:

The bag of words is created by removing the repetition. It is created by the following code:

```
bag_of_words = set(words)
```

5. Calculating Word Frequency:

The word frequency holds the frequency of the words.

Word	Frequency
w_1	$freq(w_1)$
w_2	$freq\ (w_2)$
w_3	$freq\ (w_3)$
…	…
w_n	$freq(w_n)$

Fig 5: Representation of Word Frequency

Where: $freq(w_i)$ is the frequency of the word (w_i). It is calculated by the following code:

```
Nw = len(words)
freq = {}
for word in bag_of_words:
    freq[word] = words.count(word) / Nw
```

6. Calculating Cosine Similarity:

The cosine similarity is calculated using formula (4). It is calculated by the following code:

```
# calculate the dot product of two vectors
def dot(vector1, vector2):
    sum = 0
    for key in vector1:
        if key in vector2:
            sum += vecor1[key] * vector2[key]
    return sum

# calculate the norm of a vector
def norm(vector):
    sum = 0
    for key in vector:
        sum += vector[key]**2
    return math.sqrt(sum)

# calculate the cosine similarity
value1 = dot(freq1, freq2)
value2 = norm(freq1) * norm(freq2)
cosine = value1 / value2

print("cosine similarity =", cosine)
```

4. RESULTS AND DISCUSSION

The developed program was tested on two experimental texts from Wikipedia [14]. The program performed the steps of text similarity and provided the following results:

List of Words:

The list of words is created for the two texts. It is shown in the following view:

```
List of Words (Text 1):
0  abstract
1  analogous
2  available
3  based
4  chooses
5  clinical
...

List of Words (Text 2):
0  abstraction
1  abstractive
2  abstractive
3  abstractive
4  applied
5  apply
...
```

Bag of Words:

The bag of words is created for the two texts. It is shown in the following view:

```
Bag of Words (Text 1):
0  abstract
1  analogous
2  available
3  based
4  chooses
5  clinical
...

Bag of Words (Text 2):
0 abstraction
1 abstractive
```

```
2 applied
3 apply
4 based
5 build
...
```

Word Frequency:

The word frequency is calculated for the two texts. It is shown in the following view:

```
Word Frequency (Text 1):
abstract        : 0.013513513513513514
analogous       : 0.013513513513513514
available       : 0.013513513513513514
based           : 0.013513513513513514
chooses         : 0.013513513513513514
clinical        : 0.013513513513513514
...

Word Frequency (Text 2):
abstraction     : 0.013513513513513514
abstractive     : 0.04054054054054054
applied         : 0.013513513513513514
apply           : 0.013513513513513514
based           : 0.013513513513513514
build           : 0.013513513513513514
...
```

Cosine Similarity:

The cosine similarity is calculated for the two texts. It is shown in the following view:

```
Cosine Similarity = 0.3467255099282032
```

The resulting cosine value shows that the percentage of similarity between the two texts is about (35%).

5. CONCLUSION

Text similarity is one of the important applications of machine learning. It is used to measure the similarity between texts and determine if they are similar or not. Text similarity is used a lot in many applications, for example: searching, ranking, clustering, classification, and recommendation.

In this research, the researcher developed a program in Python to measure the similarity between texts using cosine similarity method. The developed program performed the basic steps of text similarity: preprocessing text, word-tokenization, creating list of words, creating bag of words, calculating word frequency, and calculating cosine similarity.

The program successfully provided the required results: list of words, bag of words, word frequency, and cosine similarity. The resulting cosine value shows the percentage of similarity between the texts.

In the future, more work needs to be done to examine the cosine similarity method in other languages such as Arabic.

6. REFERENCES

[1] Hotho, A., Nürnberger, A., & Paass, G. (2005). "A Brief Survey of Text Mining". *LDV Forum - GLDV Journal for Computational Linguistics and Language Technology*. 20, 19-62.

[2] Salton, G. & Lesk, M. E. (1965). "The SMART Automatic Document Retrieval Systems: An Illustration". *Communications of the ACM*. 8 (6): 391-398.

[3] Salton, G. (1971). "The SMART Retrieval System: Experiments in Automatic Document Retrieval". Englewood Cliffs, N.J.: Prentice Hall Inc.

[4] Salton, G., Wong, A., & Yang, C. (1975). "A Vector Space Model for Automatic Indexing". *Communications of the ACM*, 18(11), 613-620.

[5] Salton, G., & Buckley, C. (1988). "Term-Weighting Approaches in Automatic Text Retrieval". *Information Processing and Management*, 24(5), 513-523.

[6] Salton, G. & McGill, M. (1983). "Introduction to Modern Information Retrieval". McGraw Hill Book Co, New York.

[7] Salton, G., Allan, J., & Buckley, C. (1994). "Automatic Structuring and Retrieval of Large Text Files". *Communications of the ACM*, 37(2), 97-108.

[8] Python: https://www.python.org

[9] Numpy: https://www.numpy.org

[10] Pandas: https:// pandas.pydata.org

[11] Matplotlib: https://www. matplotlib.org

[12] NLTK: https://www.nltk.org

[13] SciKit: https://scikit-learn.org

[14] Wikipedia: https://en.wikipedia.org