Text Similarity using NLP

by
Dhruv Bohara

TE-Computer Engineering

Synopsis

Introduction

Text similarity plays an increasingly important role in text related research and applications in tasks such as information retrieval, text classification, document clustering, topic detection, topic tracking, questions generation, question answering, essay scoring, short answer scoring, machine translation, text summarization and others. Finding similarity between words is a fundamental part of text similarity which is then used as a primary stage for sentence, paragraph and document similarities. Words can be similar in two ways lexically and semantically. Words are similar lexically if they have a similar character sequence. Words are similar semantically if they have the same thing, are opposite of each other, used in the same way, used in the same context and one is a type of another.

For instance, how similar are the phrases ***"the cat ate the mouse"*** with *"**the mouse ate the cat food"*** by just looking at the words?

- On the surface, if you consider only word level similarity, these two phrases appear very similar as 3 of the 4 unique words are an exact overlap. It typically does not take into account the actual meaning behind words or the entire phrase in context.
- Instead of doing a word for word comparison, we also need to pay attention to context in order to capture more of the semantics. To consider semantic similarity we need to focus on phrase/paragraph levels (or lexical chain level) where a piece of text is broken into a relevant group of related words prior to computing similarity. We know that while the words significantly overlap, these two phrases actually have different meanings.

Motivation

We might ask to ourselves, what is the need for text similarity? Why do we want to know if any two texts are similar? Well, we always need to compute the similarity in meaning between texts. Here are some applications of text similarity:

- Search engines need to model the relevance of a document to a query, beyond the overlap in words between the two. For instance, question-and-answer sites such as Quora or Stackoverflow need to determine whether a question has already been asked before.
- In legal matters, text similarity task allow to mitigate risks on a new contract, based on the assumption that if a new contract is similar to a existent one that has been proved to be resilient, the risk of this new

contract being the cause of financial loss is minimised. Here is the principle of Case Law principle. Automatic linking of related documents ensures that identical situations are treated similarly in every case. Text similarity foster fairness and equality. Precedence retrieval of legal documents is an information retrieval task to retrieve prior case documents that are related to a given case document.

- In customer services, AI system should be able to understand semantically similar queries from users and provide a uniform response. The emphasis on semantic similarity aims to create a system that recognizes language and word patterns to craft responses that are similar to how a human conversation works. For example, if the user asks "What has happened to my delivery?" or "What is wrong with my shipping?", the user will expect the same response.

Method Used: Cosine Similarity and Word Embeddings

In order to perform text similarity using NLP techniques, these are the standard steps to be followed:

- **Text Pre-Processing*:***

In day to day practice, information is being gathered from multiple sources be it web, document or transcription from audio, this information may contain various types of garbage values, noisy text, encoding. This needs to be cleaned in order to perform further tasks of NLP. In this preprocessing phase, it should include removing non-ASCII values, special characters, HTML tags, stop words, raw format conversion and so on.

- **Feature Extraction:**

To convert the text data into a numeric format, text data needs to be encoded. Various encoding techniques are widely being used to extract the word-embeddings from the text data such techniques are bag-of-words, TF-IDF, word2vec.

- **Vector Similarity:**

Once we will have vectors of the given text chunk, to compute the similarity between generated vectors, statistical methods for the vector similarity can be used. Such techniques are cosine similarity, Euclidean distance, Jaccard distance, word mover's distance. Cosine similarity is the technique that is being widely used for text similarity.

- **Decision Function:**

From the similarity score, a custom function needs to be defined to decide whether the score classifies the pair of chunks as similar or not. Cosine similarity returns the score between 0 and 1 which refers 1 as the exact similar and 0 as the nothing similar from the pair of chunks. In regular practice, if the similarity score is more than 0.5 than it is likely to similar at a somewhat level. But, this can vary based on the application and use-case.

Implementation Steps for Text Pre Processing:

- **Data Preparation:**
  Here, we have considered the list of questions and prepared the semantically similar questions for those. We have considered all the possible pair of questions with their actual labels which will define whether the pair of questions are similar or not.

- **Data Pre-Processing:**
  Once we had prepared the pair of questions with their actual labels, we have performed text pre-processing techniques in order to clean the text for the further task.

- **Uniform Case:**
  For uniformity of case, all the sentences are converted to lower case.

- **Remove Stop Words:**
  The most widely used library for pre-processing is NLTK and the list of stop-words provided by them are ‘the’, ‘is’, ‘are’, ‘a’, ‘an’, and so on.
  This can be done by installing the NLTK library.
  The words like ‘no’, ‘not’, etc are used in a negative sentence and useful in semantic similarity. So before removing these words observed the data and based on your application one can select and filter the stop words.

- **Remove Punctuation:**
  Punctuation characters are $, “, !, ?, etc. Python string class provides the list of punctuation. We are removing punctuation because they are not providing any information related to semantic similarity.

- **Remove Non-Ascii Characters:**
  Like punctuation, non-ASCII characters are not useful to capture semantic similarity.

A Sample Code for PreProcessing:

```
import nltk
from nltk import word_tokenize
from nltk.corpus import stopwords
from unidecode import unidecode
import stringdef pre_process(corpus):
    # convert input corpus to lower case.
    corpus = corpus.lower()
    # collecting a list of stop words from nltk and
punctuation form
    # string class and create single array.
    stopset = stopwords.words('english') +
list(string.punctuation)
    # remove stop words and punctuations from string.
    # word_tokenize is used to tokenize the input corpus in
word tokens.
   corpus = " ".join([i for i in word_tokenize(corpus) if i
not in stopset])
    # remove non-ascii characters
    corpus = unidecode(corpus)
    return corpuspre_process("Sample of non ASCII: Ceñía.
How to remove stopwords and punctuations?")
```

Output:

```
'sample non ascii cenia remove stopwords punctuations'
```

Lemmatization:

Lemmatization is the process of producing morphological variants of a root/base word of the language. The root word is called lemma. A Lemmatization algorithm reduces the words ‘chocolates’ to the root word ‘chocolate’. NLTK library provides a WordNetLemmatizer

Sample Code for Lemmatization:

```
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenizelemmatizer =
WordNetLemmatizer()sentence = "The striped bats are hanging
on their feet for best"words = word_tokenize(sentence)for w
in words:
    print(w, " : ", lemmatizer.lemmatize(w)
```

Output:

```
The                                 :                                  The
striped                             :                              striped
bats                                 :                                  bat
are                                 :                                  are
hanging                             :                              hanging
on                                  :                                   on
their                               :                                 their
feet                                :                                  foot
for                                 :                                  for
best                                :                                 best
```

Feature Extraction:

The Features are the representation of a sequence of words or sentence in the numeric vector. Using different word embeddings we can represent the same sentence differently in numbers. Here we will use TF-IDF.(Term Frequency – Inverse Document Frequency)
Using TF-IDF embeddings, word will be represented as a single scaler number based on TF-IDF scores. TF-IDF is the combination of TF (Term Frequency) and IDF (Inverse Document Frequency). TF gives the count of word t in document d. Mathematically we can write tf(t,d). IDF gives information about how the word is common or rare across all document. It is the logarithmically scaled inverse fraction of the documents that contain the word.

Mathematically,

idf(t,D) = log(N/dfi) where,
N or |D| = Total Number of Document
dfi = Number of document where the term t appears.

TF-IDF(t, d, D) = tf(t,d) . idf(t,D)

The scikit-learn library provides easy implementation of TF-IDF.

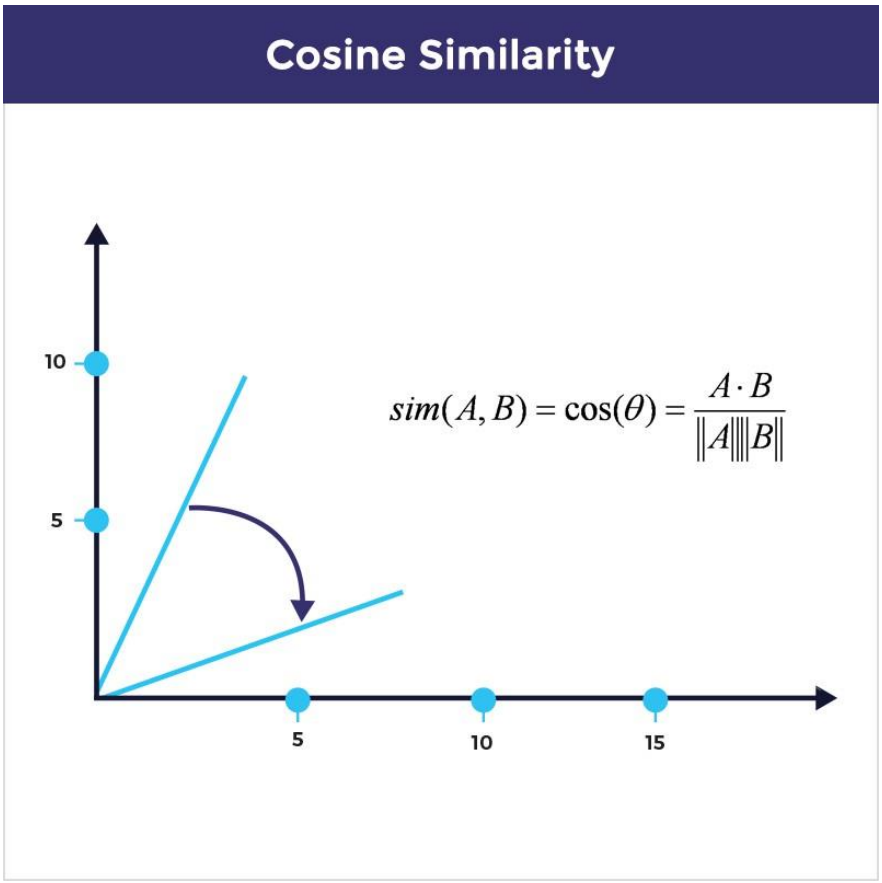
```
from sklearn.feature_extraction.text import
TfidfVectorizer# sentence paircorpus = ["A girl is styling
her hair.", "A girl is brushing her hair."]for c in
range(len(corpus)):
    corpus[c] = pre_process(corpus[c])# creating vocabulary
using uni-gram and bi-gram
tfidf_vectorizer = TfidfVectorizer(ngram_range=(1,2))
tfidf_vectorizer.fit(corpus)feature_vectors =
tfidf_vectorizer.transform(corpus)
```

So, after applying TF-IDF, vectors are generated for the corresponding words and then these vectors are trained by our model to give us an output.

Vector Similarity:

The generated word embeddings need to be compared in order to get semantic similarity between two vectors. We use the method of Cosine Similarity for these vector comparison.

Cosine Similarity:
It is the most widely used method to compare two vectors. It is a dot product between two vectors. We would find the cosine angle between the two vectors. For degree 0, cosine is 1 and it is less than 1 for any other angle.



The closer the value is to 1, more similar are the vectors.

This is a python code which uses cosine similarity to measure the similarity between two sentences:

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# X = input("Enter first string: ").lower()
# Y = input("Enter second string: ").lower()
X ="I love horror movies"
Y ="Lights out is a horror movie"

# tokenization
X_list = word_tokenize(X)
Y_list = word_tokenize(Y)

# sw contains the list of stopwords
sw = stopwords.words('english')
l1 =[];l2 =[]

# remove stop words from the string
X_set = {w for w in X_list if not w in sw}
Y_set = {w for w in Y_list if not w in sw}

# form a set containing keywords of both strings
rvector = X_set.union(Y_set)
for w in rvector:
    if w in X_set: l1.append(1) # create a vector
    else: l1.append(0)
    if w in Y_set: l2.append(1)
    else: l2.append(0)
c = 0

# cosine formula
for i in range(len(rvector)):
        c+= l1[i]*l2[i]
cosine = c / float((sum(l1)*sum(l2))**0.5)
print("Similarity: ", cosine)
```

Output:

```
Similarity: 0.2886751345948129
```

Conclusion

In this project synopsis, we reflected on text similarity as a foundational technique for a wide range of tasks. We argued that while similarity is well grounded in psychology, text similarity is less well-defined. We used the technique of embedded texts to preprocess our data to make the sentences more comparable and remove the details which are not useful in comparison. We then apply feature extraction using TF-IDF technique to generate the words in the form of vectors. These vectors are then compared using cosine similarity to provide us the similarity score between the texts. Thus, we find out that using TF-IDF and cosine similarity, we can compare the texts with great confidence.

References:

https://www.aclweb.org/anthology/R11-1071.pdf

https://medium.com/@Intellica.AI/comparison-of-different-word-embeddings-on-text-similarity-a-use-case-in-nlp-e83e08469c1c

https://www.machinelearningplus.com/nlp/cosine-similarity/

https://medium.com/@adriensieg/text-similarities-da019229c894

https://www.geeksforgeeks.org/python-measure-similarity-between-two-sentences-using-cosine-similarity/

https://towardsdatascience.com/overview-of-text-similarity-metrics-3397c4601f50

Youtube