International Journal of Scientific Research in Computer Science, Engineering and Information Technology
ISSN : 2456-3307 (www.ijsrcseit.com)
doi : https://doi.org/10.32628/CSEIT2390133

A Survey of Numerous Text Similarity Approach

Joyinee Dasgupta, Priyanka Kumari Mishra, Selvakuberan Karuppasamy , Arpana Dipak Mahajan and Subhashini Lakshminarayanan
Advanced Technology Centers, India

Article Info
Publication Issue :
Volume 9, Issue 1
January-February-2023

Page Number : 184-194

Article History
Accepted: 01 Feb 2023
Published: 11 Feb 2023

ABSTRACT

One of the most common NLP use cases is text similarity. Every domain comes with a variety of use cases. The most common uses of text similarity include finding related articles/news/genres, efficient use of search engines, classification of related issues on any topic, etc. It serves as a framework for many text analytics use cases. Methods to solve text similarity use cases have been around for a while, but the main drawbacks of the old methods are loss of dependency information, difficulty remembering long conversations, exploding gradient problems, etc. Recent advanced deep learning-based models pay attention to both contiguous and distant words, making their learning ability more rigorous. This white paper focuses on various text similarity techniques that can be used in everyday life to solve these use cases.

Keywords : Natural Language Processing; Euclidian distance, Cosine similarity, Jaccard Distance, word embeddings, Language Models ,Universal Sentence Encoders

I. INTRODUCTION

Natural Language Processing is one of the fields that has advanced the most recently and has assisted us in resolving a wide range of issues in many sectors of society. It consists of text categorization, text similarity, text production, text summarization, machine translation, chatbots, information retrieval systems, and other related technologies. The majority of industries are currently dealing with the issue of text similarity. Data preparation, which involves cleaning the data by removing noise from the data and undesirable characters, is a necessary step in the process of text similarity or any natural language processing. It leverages advanced mathematical concepts like vector, trigonometry, linear algebra etc to calculate the similarity between two sentences . Similarity measures are used in a wide range of applications, including automated document linking, information retrieval systems, paraphrase detection, search engines, text classification etc.

II. LITERATURE SURVEY

This study examines how semantic similarity techniques have changed throughout the years, separating them according to the underlying methodologies that underlie them. The measurement of semantic equivalency amongst of text is known as Semantic Textual Similarity (STS)[1]. Instead of a

Copyright: © the author(s), publisher and licensee Technoscience Academy. This is an open-access article distributed under the terms of the Creative Commons Attribution Non-Commercial License, which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited

simple yes or no answer, semantic similarity algorithms typically provide a ranking or percentage of similarity across texts. Semantic relatedness and semantic similarity are sometimes used interchangeably [2]. The goal of this review is to present a thorough analysis of the different semantic similarity methodologies, including the most recent developments based on deep neural techniques. Text classification approaches have been widely employed to speed up multimedia data processing in numerous multimedia applications, such as video/image tagging and multimedia recommendation [3]–[7].

In this study [8], the Jaccard and Cosine similarity metrics for measuring text similarity are compared. The text similarity was measured by the Jaccard similarity index. The text was converted into a vector space model along with their distance is calculated utilizing the "Word2Vec" method in the Cosine similarity algorithm.

Kaundal et al. reviewed two methods for calculating short text semantic similarity (STSS), a vector space model and a knowledge-based model that used WordNet [7]. In [9]Three approaches have been used to discuss the existing studies on text similarity: string-based, knowledge-based, and corpus-based similarity. Each method is based on a distinct perspective, and they all measure how closely two short texts are related. A definitive perspective on this subject is also provided by the introduction of datasets, which are frequently used as benchmarks for assessing approaches in this area. The utilisation of methods that incorporate several viewpoints yields better outcomes. In [10], a small gap between those traits suggests a high level of similarity, whereas a big distance suggests a low level of similarity . Few of the distance metrics used in calculating document similarity are Euclidean distance, Cosine distance, and Jaccard coefficient metrics.

III. RESULTS AND DISCUSSION

A. Perception of Similarity

Similarity is the measurement of how similar or how different a set of things look like. In other words, it is a metric that helps to decide whether two objects are how much same or how much different from each other. If the distance between two objects is large it shows their dissimilarity and vice versa. This metric generally lies from 0 to 1;0 being high dissimilar and 1 being highly similar. Similarity is a very metric and it is highly dependent on domain. Let's say if we want to compare two laptops with same colour, screen size, positioning of the keyboard, webcam, type of charging point and socket used etc. We might call it similar if the features of comparison are almost same but if the features for comparison for the use case is different like may be the OS used, the memory, presence of graphic card or the speed etc are different we could say for the same laptop its dissimilar. So, the perception of similarity or dissimilarity completely depends on the use case and the business objective. So, need to be very clear when measuring similarity.

Although the concept may seem straightforward, similarity is the cornerstone of many machine learning approaches. For instance, similarity is used by the K-Nearest-Neighbors classifier to categorize new entity or, and by K-means clustering to allocate data points to suitable groups. Even recommendation algorithms use neighborhood-based collaborative filtering techniques that identify a user's neighbours based on similarity.

Example: -

Let's take an example of 2 very simple sentences.

1. I am hungry.
2. I want to eat something.

As a human when we read through the sentences, we can understand that these two sentences are similar but for a machine it's very difficult to

understand the context though both the sentences are exactly same. Let's get into the various similarity techniques to measure it .

B. Measures

1. Text Distance

It offers a description of the semantic similarity among two words based on distance. Length, distribution, and semantic distance are the 3 ways to measure distance.

- **Length Distance**

By computing the distance length of vector text using the text's numerical properties, length distance has been used to measure text similarity.

- **Euclidean Distance**

The Pythagoras rule is used to determine the separation between two points. The similarity score decreases and vice versa when the distance between two vectors increases. The output of the Euclidean distance lies from 0 to infinity. It's very difficult to state whether its similar or dissimilar so normalisation is a way to convert the score between 0 to 1 so that we can measure the distance between a pair of sentences. Using python to calculate the Euclidean distance.

```
In [2]: def squared_sum(x):
   ...:   """ return 3 rounded square rooted value """
   ...:
   ...:   return round(sqrt(sum([a*a for a in x])),3)
   ...:
   ...: def euclidean_distance(x,y):
   ...:   """ return euclidean distance between two lists """
   ...:
   ...:   return sqrt(sum(pow(a-b,2) for a, b in zip(x, y)))
   ...:
   ...: embeddings = [nlp(sentence).vector for sentence in sentences]
   ...: distance = euclidean_distance(embeddings[0], embeddings[1])
   ...: print(distance)
   ...: def distance_to_similarity(distance):
   ...:   return 1/exp(distance)
   ...: distance_to_similarity(distance)
   ...: # OUTPUT
4.328308287052498
Out[2]: 0.013189842053339177
```

Initially when the Euclidean distance of 4.32 is calculated, it's difficult to state whether the pair of sentences is similar or dissimilar. So, we normalise it using the Euler's constant (1/e to the power distance).

- **Cosine Similarity**

Cosine similarity, which is the cosine of the angle between the two vectors, is used to calculate how similar two vectors are to one another. When two vectors nearly point in the same direction, it can identify this. The cosine similarity is 1 if the angle between the vectors is 0 degrees.

Using python to calculate the cosine similarity:

```
In [3]: def cos_similarity(x,y):
   ...:   """ return cosine similarity between two lists """
   ...:
   ...:   numerator = sum(a*b for a,b in zip(x,y))
   ...:   denominator = squared_sum(x)*squared_sum(y)
   ...:   return round(numerator/float(denominator),3)
   ...:
   ...: cos_similarity(embeddings[0], embeddings[1])
   ...:
   ...: # OUTPUT
Out[3]: 0.548
```

Sentence similarity resulted in a slightly better score. Jaccard similarity does not work with text embeddings, so it is rarely used when working with text data. This means that it is limited to assessing the lexical similarity of texts. H. Degree of similarity of documents at the word level. The difference between cosine and Euclidean metrics is that cosine similarity is not affected by feature vector size or length. A sparse vector of text embeddings makes Euclidean distance a problem. Therefore, cosine similarity is usually preferred over Euclidean distance when working with text data. The only use case for length-dependent text similarity that comes to mind is plagiarism detection.

- **Manhattan Distance**

According to the Manhattan distance metric, the separation between two locations is equal to the total of their absolute differences in Cartesian coordinates. It is the total sum of the difference between the x- and y-coordinates, to put it simply.

```
In [19]: from sklearn.feature_extraction.text import CountVectorizer
    ...:
    ...: document = ["I am hungry",
    ...:             "I donot want to eat"]
    ...:
    ...: # Create a Vectorizer Object
    ...: vectorizer = CountVectorizer()
    ...:
    ...: vectorizer.fit(document)
    ...:
    ...: # Printing the identified Unique words along with their indices
    ...: print("Vocabulary: ", vectorizer.vocabulary_)
    ...:
    ...: # Encode the Document
    ...: vector = vectorizer.transform(document)
    ...: a=vector.toarray()
    ...:
    ...: def manhattan_distance(point1, point2):
    ...:     return sum(abs(value1 - value2) for value1, value2 in zip(point1,
point2))
    ...: manhattan_distance(a[0],a[1])
Vocabulary:  {'am': 0, 'hungry': 3, 'donot': 1, 'want': 5, 'to': 4, 'eat': 2}
Out[19]: 6
```

Here the number 6 signifies the distance between two sentences and it fails to capture context and more on string.

- **Hamming Distance**

The distance between two chains of equal length is determined by the number of locations where the linked symbols diverge. In other words, it determines the minimal number of mistakes that can occur when converting one string to another or the minimal number of substitutions needed.

- **Distribution Distance**

There are majorly two downsides to using length distance to assess similarity:
When query Q is used to obtain response A, the associated similarity is not symmetrical, but it would be adequate for symmetrical problems like
Sim (A, B) = Sim (B, A).
Secondly, a danger is involved there in utilizing distance and length to gauge similarity measurement without understanding statistical properties of the data. When establishing if two articles come from the same distribution, the distribution distance is employed to measure how similar the two papers are. We briefly discuss many distributions distance approaches

- **kullback-leibler-divergence**

For a given random variable or set of occurrences, KL divergence is a metric for assessing the relative differences between two probability distributions. Relative entropy is another name for KL divergence.

- **Jensen-Shannon-divergence**

The degree to which the label distributions of several facets diverge entropically from one another is gauged by the Jensen-Shannon divergence (JS). It is symmetric and is based on the Kullback-Leibler divergence. The lesser the JS distance, similar the distribution of the two documents would be similar. The Jensen-Shannon divergence is calculated as follows:

$$JS = ½ * [KL (P a || P) + KL (P d || P)]$$

- **Semantic Distance**

The idea of distance between objects is based on the similarity of their meaning or semantic content rather than lexicographical similarity in the case of semantic distance, which is a distance metric defined over a set of texts or phrases. These are mathematical instruments that are used to gauge the strength of the semantic connection between language units, concepts, or instances by a numerical description derived from a comparison of the data evidencing their meaning or outlining their characteristics.

- **Word Mover's Distance**

WMD makes use of outcomes of cutting-edge embedding methods like Glove, Word2vec, which produce word-embeddings of extraordinary quality and scale organically to extremely huge data sets. These embedding methods show how word vector operations often preserve semantic links.
It treats text documents as such and makes use of the weighted point cloud of embedded words that word vector embeddings contain. The two documents are

separated by the smallest total distance that words from text document A must cover to precisely match the point cloud in text document B.

2. Text Representation

Both lexically and thematically, texts may be similar to one another. Words are lexically comparable if their character patterns are similar across the text's words. Two words are semantically similar if they have the same meaning, oppose one another, are employed in the same way, are used in the same context, or are examples of one another. Lexical similarity is evaluated with a variety of text representation metrics, whilst semantic similarity is introduced with the help of the string-based methodology, corpus-based method, semantic text matching, and graph-structure-based approach.

- **String Based**

When two text string are compared or approximate string matched, string similarity metrics are used to determine how similar or dissimilar (distance) two text strings are. The most common string similarity metrics used in the symmetric package are represented in this survey.

- **Phrase Based**

The phrase-based method's fundamental building block is a phrase word, and its primary approaches include the dice coefficient, Jaccard, and others.

- **Dice**

The definition of Dice's coefficient is two times the number of terms that are shared by both strings divided by all of the terms in both strings.

```
In [15]: def compute_dice_sim(str1, str2):
    ...:     a = set(str1.split())
    ...:     b = set(str2.split())
    ...:     c = a.intersection(b)
    ...:     return 2*float(len(c)) / (len(a) + len(b))
    ...:
    ...:
    ...: sent1 = "I am hungry."
    ...: sent2 = "I want to eat something."
    ...:
    ...: dice = compute_dice_sim(sent1, sent2)
    ...: print(dice)
0.25
```

- **Jaccard Index**

A Jaccard index, commonly known as the Jaccard similarity coefficient, treats data elements as a set. By dividing size of union by size of the intersection of the two sets, this can be calculated. Consider the same illustration.

Sentence 1: I am hungry.

Sentence 2: I want to eat something.

We will first do text normalisation to remove word roots and lemmas before computing the similarity using the Jaccard similarity. In the case of our example sentences, there are no words to eliminate, thus we can proceed to the following section. Python function for Jaccard similarity:

```
In [3]: def jaccard_similarity(x,y):
   ...:     """ returns the jaccard similarity between two lists """
   ...:     intersection_cardinality = len(set.intersection(*[set(x), set(y)]))
   ...:     union_cardinality = len(set.union(*[set(x), set(y)]))
   ...:     return intersection_cardinality/float(union_cardinality)
   ...:
   ...: sentences = ["I am hungry.",
   ...: "I want to eat something."]
   ...: sentences = [sent.lower().split(" ") for sent in sentences]
   ...: jaccard_similarity(sentences[0], sentences[1])
   ...: #Output
Out[3]: 0.14285714285714285

In [4]:
```

- **Character Based**

To express the similarity between two texts, character-based similarity calculations are based on the similarity of characters within the text. LCS (Longest Common Substring), Jaro similarity, edit distance and other related techniques are introduced.

- **LCS**

A typical illustration of a character-based similarity measure is LCS. The longest continuous chain of characters that can be found in both strings is taken into account by the Longest Common SubString (LCS) algorithm.

```
In [11]: def lcs(X, Y, m, n):
    ...:
    ...:     if m == 0 or n == 0:
    ...:        return 0;
    ...:     elif X[m-1] == Y[n-1]:
    ...:        return 1 + lcs(X, Y, m-1, n-1);
    ...:     else:
    ...:        return max(lcs(X, Y, m, n-1), lcs(X, Y, m-1, n));
    ...:
    ...:
    ...:
    ...: # Driver program to test the above function
    ...: X = "AGGTAB"
    ...: Y = "GXTXAYB"
    ...: print ("Length of LCS is ", lcs(X, Y, len(X), len(Y)))
Length of LCS is  4
```

- **Edit Distance**

Levenshtein distance, also known as edit distance, is a metric used to compare the similarity of two strings, known as source and target strings. The number of editing operations (deletions, insertions, or substitutions) required to transform a source string into a target string is called the distance between the source and target strings. The similarity between two strings increases as the distance between them decreases.

```
In [17]: import nltk
    ...:
    ...: sent1 = "It might help to re-install Python if possible."
    ...: sent2 = "It can help to install Python again if possible."
    ...: sent3 = "It can be so helpful to reinstall C++ if possible."
    ...: sent4 = "help It possible Python to re-install if might." # This has the same words as sent1 with a
different order.
    ...: sent5 = "I love Python programming."
    ...:
    ...: ed_sent_1_2 = nltk.edit_distance(sent1, sent2)
    ...: ed_sent_1_3 = nltk.edit_distance(sent1, sent3)
    ...: ed_sent_1_4 = nltk.edit_distance(sent1, sent4)
    ...: ed_sent_1_5 = nltk.edit_distance(sent1, sent5)
    ...:
    ...:
    ...: print(ed_sent_1_2, 'Edit Distance between sent1 and sent2')
    ...: print(ed_sent_1_3, 'Edit Distance between sent1 and sent3')
    ...: print(ed_sent_1_4, 'Edit Distance between sent1 and sent4')
    ...: print(ed_sent_1_5, 'Edit Distance between sent1 and sent5')
14 Edit Distance between sent1 and sent2
19 Edit Distance between sent1 and sent3
32 Edit Distance between sent1 and sent4
33 Edit Distance between sent1 and sent5
```

- **Jaro Similarity**

The measure of resemblance between two strings is called Jaro Similarity. The Jaro distance has a value between 0 and 1. where 0 indicates that there is no resemblance between the two strings and 1 indicates that they are equal.

- **Corpus Based**

Corpus-based similarity analyses information from a huge corpus to determine the semantic similarity between terms. The corpus-based method calculates text similarity using data from the corpus; this data may be either have linguistic characteristic or a likelihood of co-occurrence.

- **Bag-of-words model**

The fundamental tenet of the bag-of-words method is to analyse a document as a collection of words without considering their usage order. The most often used word bag-based techniques are LSA (latent semantic analysis), TF-IDF (term frequency-inverse document frequency), and BOW (bag of words).

- **TF-IDF**

Each text will be transformed into its vector representation via the TF-IDF vectorizer. As a result, we will be able to approach each text as a collection of points in a multidimensional space.

- **Shallow Window-Based Methods**

The shallow window-based techniques differ from the word bag model in significant ways, one of which being the semantic distance between words. This isn't considered in the bag of words model. Low-dimensional real vectors, on the other hand, can be learned in unstructured text without a mark by creating word vectors using shallow window-based techniques, which spatially cluster similar words together.

- **GloVe**

Glove is a global vector method that blends count-based methods with direct prediction techniques like word2vec (such as PCA, principal component

analysis). With the help of GloVe's global matrix factorization method, a matrix that shows whether or not a document contains words is produced. The GloVe model analyses the word vector such that the dot product of the words equals the logarithm of the likelihood of those words occurring concurrently. The Glove Model can be written down as follows. (J. Pennington, R. Socher, and C. Manning).

Using statistics to determine the relationship between words is the core premise of the GloVe word embedding. In contrast to the occurrence matrix, the co-occurrence matrix lets you know how frequently a particular word pair appears with another. Each value in the co-occurrence matrix represents a word pair that commonly appears together.

- **Bert**

The popular attention model Transformer's bidirectional training for language modeling has been translated into BERT's main technological advancement. In order to determine the contextual links between words (or subwords) in a document, BERT uses the Transformer attention mechanism. Transformer's basic configuration consists of two separate processes: a text input encoder and a job prediction decoder.

- **Word2Vec**

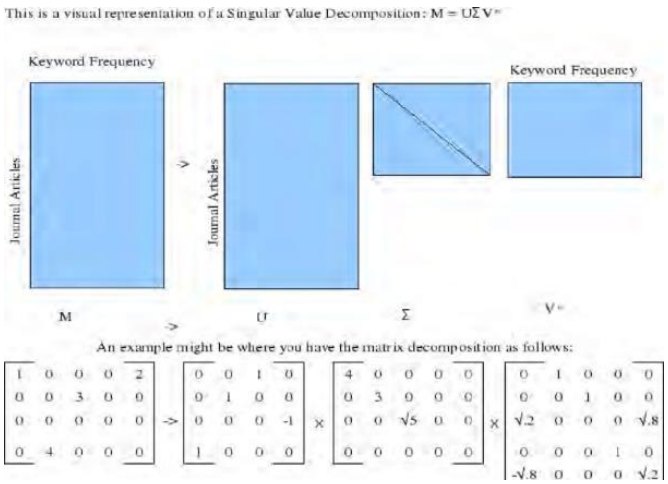
In the Word2Vec model, words are converted into vectors. You can then use the cosine similarity formula to calculate the similarity value from the word vector data that the Word2Vec model has provided. There are two pre-trained models for Word2Vec: the continuous word bag model called CBOW, and the skip-gram model.As an illustration, consider the CBOW model, which consists of the input, mapping, and output layers and predicts the intermediate words that depends on context. The specific heat vector for the inter-word context is the first input the model gets. The shared input weight matrix W is then multiplied by each unique heat vector. The word vectors' hidden layer vector is then updated to include the average. The final probability distribution is produced by multiplying the hidden layer vector by the weight matrix W of the output and then applying SoftMax algorithm to it.

- **Matrix Factorization Method**

Matrix factorization is complex mathematical calculation that discover latent features which tells about the interactions among entities/texts etc. Mainly there are different methods to calculate text similarity using matrix factorization methods.

- **LSA**

A method for constructing a vector representation of a document is latent semantic analysis. When a document is represented as a vector, it is possible to compare documents for similarity by figuring out how far off the vectors are from one another.

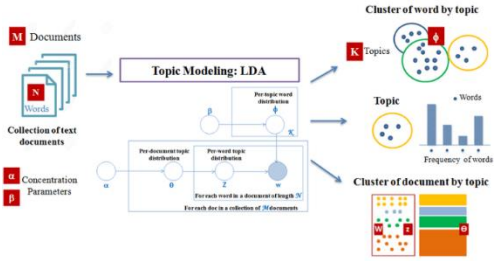


Reference -viewcontent.cgi.pdf

- **LDA**

The unsupervised technique Latent Dirichlet Allocation (LDA) gives each document a value for each specified subject (let's suppose we choose to

search for 5 distinct themes in our corpus). While Dirichlet is a sort of probability distribution, latent is another synonym for hidden (i.e., properties that cannot be assessed directly). LDA views each document as a combination of words and topics, and vice versa. Each term and each subject are covered one by one. Each word will be given a subject at random, and the frequency with which it appears in that topic and with which other terms will be evaluated. As a result, it is a particularly popular tool for identifying text similarities since it groups related words, documents, or phrases.



Reference-
Multilingual_topic_modelling_for_tracking_COVID-19.pdf

- **Knowledge graph**

A graph kernel compares graph substructures to determine how similar two graphs are to one another. It allows for the consideration of structural information in text for determining document similarity. A graph kernel's ability to calculate similarity depends on the data that is represented in the graphs. A weighted co-occurrence graph serves as the first representation of the text document. Then, using a similarity matrix based on word similarities, it is turned into an enhanced document network by automatically constructing related nodes and edges (or relationships). Matching terms and patterns contribute to document similarity based on their relevance since a supervised term weighting method is employed to weight the terms and their relationships. The similarity measure can go beyond exact term and association matching thanks to graph enrichment. We utilise the data in the enriched weighted graphs to calculate the similarity between text texts using an edge walk graph kernel. The kernel function takes two weighted co-occurrence graphs as input and outputs a similarity score based on how closely the relevant text in the two documents matches.

- **Graph Neural Network**

To design a graph neural network where there are many tiers of data and connections, the model must be utilised to specify the hierarchical relationship of the data (GNNs). The graph neural network (GNN), a connectionism model, provides an example of a graph's dependence on message transmission between its nodes. A graph neural network keeps a state that can represent information from its neighbourhood at any depth, in contrast to a normal neural network. Relationships between words can be automatically discovered, generated, and then reconstructed during processing using a special kind of GNN called a WRGNN.

- **Semantic Text Matching**

In NLP, semantic matching techniques compare two utterances to see whether their meanings are similar. Semantic text matching is the process of comparing the semantic similarity of source and destination text fragments. Based on LSA, deep learning extracts the hierarchical semantic structure from the query and content. In this case, a new expression is produced as a result of the text being encoded to extract features.

The four primary approaches used for single semantic text matching are architecture-I (for matching two phrases), architecture-II, convolutional latent semantic model (CDSSM), and deep structured semantic model (DSSM) (Architecture-II of convolutional matching model).

- **DSSM**

The search industry was where deep-structured semantic models (DSSM) were first applied. A semantic similarity model will be trained using click exposure records from a large number of search results, and Deep Neural Networks (DNN) will be used to represent the query and title as low-latitude semantic vectors, cosine distance will be utilised to gauge the relationship between the two vectors. The context loss of DSSM is partially mitigated by switching to CNN from DNN (Convolutional Neural Network). When deep learning first emerged, CNN and long-term memory (LSTM) were proposed, and the structures of these specific diagnosis extracts were also used to develop DSSM. The feature extraction layer, which takes the place of CNN or LSTM, has a completely different network structure.

- **ARC-1**

The DSSM model's shortcomings in recording question and document sequences and context information are addressed by using the CNN module to produce the suggested ARC-I and ARC-II. The ARC-I model is an interactive learning paradigm that is based on representational learning. The key distinction between the two models and the original DSSM model is the inclusion of convolution and pooling layers to extract sentence word.

The most significant portions of these connections are retrieved by pooling layer maxpooling after ARC-1 creates a number of combinatorial associations between nearby feature maps using convolution layers with different terms. The text representation will then be delivered to DSSM.

- **Multi-semantic Document Matching**

Significant local information is lost when complicated words are compressed into a single vector based on a single meaning. A single-granularity vector to characterize a piece of text is not fine enough on the basis of a single semantic, according to the deep learning model of document expression based on multiple semantics. We can examine local text similarities and synthesise the degree of text matching thanks to the extensive interactive work and multi-semantic expression it does before matching. The two main multi-semantic techniques are MatchPyramid and Multi-View Bi-LSTM (MV-LSTM).

- **MV-LSTM**

Three components make up MV-LSTM: First, each positional sentence representation is an individual sentence representation created using a bidirectional long short-term memory (Bi-LSTM); Second, many similarity functions combine to generate a similarity matrix or tensor via interactions between various positional sentence representations; A multilayer perceptron and k-Max pooling are used to aggregate these interactions to create the final matching score.

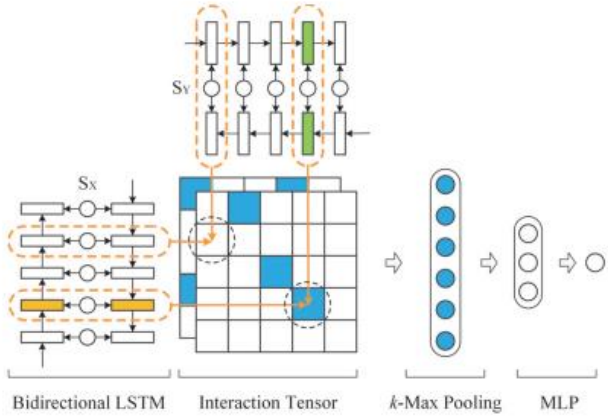


Fig: Illustration of MV-LSTM (reference - A Deep Architecture for Semantic Matching with Multiple Positional Sentence Representations (aaai.org))

- **MatchPyramid**

In this part, we introduce MatchPyramid, a novel deep architecture for text matching. By seeing the matching matrix as an image and treating text

matching as image recognition, the fundamental concept is revealed.

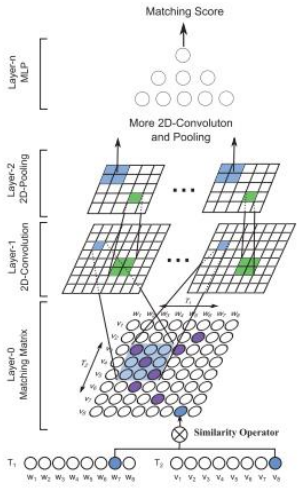


Fig: MatchPyramid on Text Matching.(reference : Text Matching as Image Recognition (aaai.org))

IV.CONCLUSION

Generally, traditional embedding methods like Word2Vec and Doc2Vec yield good results when the task only requires the broad sense of the text. On tasks like semantic text similarity or paraphrase identification, they perform better than state-of-the-art deep learning techniques, which reflects this. On the other hand, when the task necessitates something more specific than just the general meaning, such as sentiment analysis or sequence labelling, more complex contextual techniques perform better. Therefore, wherever possible, start with a short and simple process before moving on to one that requires more effort as needed.

V. REFERENCES

[1]. P. Bambroo and A. Awasthi, “LegalDB : Long DistilBERT for Legal Document Classification”.

[2]. D. Chandrasekaran and V. Mago, “Evolution of Semantic Similarity — A Survey,” vol. 54, no. 2, 2021.

[3]. X. Deng, Y. Li, J. Weng, and J. Zhang, “Feature selection for text classification: A review,” Multimed. Tools Appl., vol. 78, no. 3, pp. 3797–3816, 2019, doi: 10.1007/s11042-018-6083-5.

[4]. Z. Huang et al., Context-aware legal citation recommendation using deep learning, vol. 1, no. 1. Association for Computing Machinery, 2021. doi: 10.1145/3462757.3466066.

[5]. S. Yang, G. Huang, B. Ofoghi, and J. Yearwood, “Short text similarity measurement using context-aware weighted biterms,” Concurr. Comput. Pract. Exp., vol. 34, no. 8, pp. 1–11, 2022, doi: 10.1002/cpe.5765.

[6]. D. W. Prakoso, A. Abdi, and C. Amrit, “Short text similarity measurement methods: a review,” Soft Comput., vol. 25, no. 6, pp. 4699–4723, 2021, doi: 10.1007/s00500-020-05479-2.

[7]. A. Kaundal, “A Review on WordNet and Vector Space Analysis for Short-text Semantic Similarity,” Int. J. Innov. Eng. Technol., vol. 8, no. 1, pp. 135–142, 2017, doi: 10.21172/ijiet.81.018.

[8]. A. W. Qurashi, V. Holmes, and A. P. Johnson, “Document Processing: Methods for Semantic Text Similarity Analysis,” INISTA 2020 - 2020 Int. Conf. Innov. Intell. Syst. Appl. Proc., pp. 0–5, 2020, doi: 10.1109/INISTA49547.2020.9194665.

[9]. T. Nora Raju, P. A. Rahana, R. Moncy, S. Ajay, and S. K. Nambiar, “Sentence Similarity - A State of Art Approaches,” Proc. Int. Conf. Comput. Commun. Secur. Intell. Syst. IC3SIS 2022, pp. 0–5, 2022, doi: 10.1109/IC3SIS54991.2022.9885721.

[10]. R. Singh and S. Singh, “Text Similarity Measures in News Articles by Vector Space Model Using NLP,” J. Inst. Eng. Ser. B, vol. 102, no. 2, pp. 329–338, 2021, doi: 10.1007/s40031-020-00501-5.

Cite this article as :

Joyinee Dasgupta, Priyanka Kumari Mishra, Selvakuberan Karuppasamy , Arpana Dipak Mahajan, "A Survey of Numerous Text Similarity Approach", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 9, Issue 1, pp.184-194, January-February-2023. Available at doi :
https://doi.org/10.32628/CSEIT2390133
Journal URL : https://ijsrcseit.com/CSEIT2390133