Vectorization of Text in Natural Language Processing

[1]Parth Sethi, [2]Deval Verma
[1]Department of Mathematics, Chandigarh University, Punjab- 140413
parthsethi85@gmail.com
[2]School of Computer Science Engineering and Technology
Bennett University, Times Group
Greater Noida-201310
deval09msc@gmail.com

Abstract

(NLP) Natural language processing is the field where computers can comprehend and use natural language text or voice for beneficial purposes. Some pre-processing and feature encoding techniques are applied over unstructured text data. The text will then be transformed into numerical feature vectors so that it may be fed to computers for machine learning applications. In this work, a statistical or frequency-based word embedding techniques are used. That are Continuous Bag of Words (CBOW), (TF-IDF) Term Frequency-Inverse Document Frequency and Skip-Gram Models. This work shows a comparison of all these techniques.

Keywords: NLP, Textual data, lemmatization, stemming, TF-IDF, CBOW and Skip gram models

1. INTRODUCTION

Natural language processing helps us to deal with this important aspect of humanity. NLP is a branch of machine learning that provides computers the ability to comprehend human text and speech. For example, humans feel more connected through stories; moreover, stories are an efficient way to transmit important information from one person to another. Stories are fundamental to humans for example: “Humans like to think in tales versus facts, statistics, or calculations, and the more straightforward the story, the better.”– Yuval Noah Harari

The above quote from the book [1] clearly highlights the importance of stories for humanity. Human race is a social animal and research has shown that stories are more engaging and influencing for the brain, that is the reason why some literature is banned by the governments of various countries. Humans feel more connected through stories; moreover, stories are an efficient way to transmit important information from one person to another. Stories are fundamental to humans [1, 2, 3,4].

Some of the use cases of NLP include speech recognitions, sentiment analysis, and language translators. NLP is playing a huge role in improving the healthcare sector and it is helping in providing better results for the patients [5,6,7,8,14]. Word embedding are numerical representations of words in the shape of a low dimensional vector, we need

word embedding because machine algorithms can understand only numbers and not text or words [9,10,11,12,14]. It can take a word out of its textual context. Words that are similar will have close vectors, for instance words happy and jovial will have vectors that are akin to each other. Word embedding is also capable of capturing semantic and syntactic similarities of texts. Semantic refers to similarity of the meaning of the given texts or words and syntactic similarities refer to the similarity of words of the given texts [13,14]. There are many techniques to attain word embedding; some of the most popular techniques are listed below:

- Binary Encoding
- TF-IDF Encoding
- Word2Vec Embedding

1.1. Binary Encoding

In one binary encoding, every word which is a part of the text is represented in vector form. A word will be labelled as 0 or 1, that is how this technique gained its name. It is also known as one hot encoding. For instance, consider the following sentences “I love to read” and “I like to read”. Both sentences have semantic and syntactic similarities. Before we can encode these sentences, we must tokenize them. These two sentences will have the following vectors if we use binary encoding as our encoding technique [4].

On the left-hand side of the figures are the indices of the words. For example, look at the vector for “I like to read”, at index 0 there is the alphabet “I” so it’s one hot encoding representation is [1,0,0,0] followed by the word “like” at index 1 which will have vector [0,1,0,0]. One hot encoding technique has dimensionality problems. In the above used sentences the vocab size is extremely small, Vocab size = Number of distinctive words. So, the vocab size for the sentence “I like to read” is 4 and the same is for the sentence “I love to read.” From the above figures, we can see that most of the vectors are taken up by zeroes and single ones. Now, assume that we have a text with a vocabulary size of 10,000. Each word in that vocab will be represented by 9999 0’s and a single 1. This is not computation friendly.

It is difficult to attain syntactic and semantic information of the text if we use binary encoding. Syntactic refers to the grammatical structure of the sentence and semantic refers to the meaning of the sentence [5,6].

	I	love	read	to
0	1	0	0	0
1	0	1	0	0
2	0	0	0	1
3	0	0	1	0

	I	like	read	to
0	1	0	0	0
1	0	1	0	0
2	0	0	0	1
3	0	0	1	0

Figure 1.1. Binary encoding vectors

The above statement examples (Figure:1.1) are like each other but looking at the derived vectors we cannot obtain any information about the similarity. Binary encoding techniques cannot extract the essence of the text. We also lose the order of the words in which they appear in the text. We used sentences “I like to read” and “I love to read” but when we binary encoded those sentences words “read” and “to” lose their order, this might lead to the loss of the meaning of the entire text.

1.2. Term Frequency-Inverse Document Frequency Encoding

Term Frequency-Inverse Document Frequency is abbreviated as TF-IDF. Its score is obtained by using the following formula in (1.21):

$$tf - idf(w, d, C) = tf(w, d) * idf(w, C) \quad (1.21)$$

It is obtained by multiplying the number of times word w appears in document d with the inverse document frequency of word w in the corpus C. Most popular uses of this technique include text mining, Keyword extraction etc. Before we can understand how tf-idf works, we first must understand tf and idf, as these are multiplied to attain tf idf score [7,8] as shown in Table 1.1.

Term Frequency (TF): TF measures the frequency with which a word appears in the text. Length of the documents can greatly impact the term frequency, TF is the quantity of is directly proportional to occurrences of a word i.e., as the number of occurrences of a word increase, term frequency also increases [6]. Each document has its own term frequency. For instance, there is a high probability that a word such as "this" can appear more times in a document with 500 words as compared to the document with 50 words. To deal with these problems we perform normalization of the frequency of the words appearing in a document. This can be done by using the following formula in (1.22):

$$tf(w, d) = \frac{Number\ of\ times\ word\ w\ appears\ in\ document\ d}{Total\ number\ of\ words\ in\ the\ document} \quad (1.22)$$

For instance, our document contains 2 sentences. First sentence is "What a beautiful day", and the second sentence is "Sun is shining bright". Now, our vocabulary will look like

{"What", "a", "beautiful", "day", "Sun", "is", "shining", "bright"}

Table1.1 Values of tf (A) and tf (B)

Term	**tf(A)**	**tf(B)**
What	1/4	0
a	1/4	0
beautiful	1/4	0
day	1/4	0
Sun	0	1/4
is	0	1/4
shining	0	1/4
below	0	1/4

Inverse Document Frequency (IDF): Document frequency refers to the number of documents in a corpus that contains the term. IDF can be calculated using the following formula in (1.23).

$$IDF(w) = \int \frac{Total\ number\ of\ documents\ in\ a\ Corpus}{Number\ of\ documents\ with\ word\ w\ in\ it} \quad (1.23)$$

IDF is the measure of the significance of a word in a corpus. We need IDF because while calculating term frequency each word is given equal importance. For instance, we need to calculate the term frequency of an article about renewable sources of energy, however words such as „this", „that, „of „ may have more number of occurrences as compared to words such as „renewable", „natural" and in our case these less occurring terms holds more importance and to deal with this we will compute IDF [6]. IDF score will be less for more frequent terms and high for the rare terms. To understand IDF better, assume that we have a corpus of 5 documents and each document contains a single sentence.

Document 1 = "There might be rain today"

Document 2 = "I like to read"

Document 3 = "It may not rain today"

Document 4 = "I love to read George Orwell's work"

Document 5 = "It is such a beautiful day"

IDF score for word "I" will be calculated using these values 52 and it will be 0.39. Total number of documents in the corpus is 5 and the documents which contain the word "I" are 2. Similarly, calculate the IDF for the word "George Orwell", 51 and it will be 0.69.

Now, let's get back to tf idf. It is obtained after multiplying the term frequency with the inverse document frequency. For simplicity purposes assume we have weather corpora that contain 1000 documents and in a document of 500 words "sun" occurs 100 times. So, the term frequency of the word wills 100500 or 0.2, and "sun" is occurring in 50 documents, so the inverse document frequency will be 100050 or 1.30. The tf-idf score will be the product of tf and idf: 0.2 * 1.30 = 0.26.

1.3. Word2Vec

Word2vec converts a word into a vector, and various arithmetic operations can be performed on these vectors; it's a multilayer neural network model [13]. Similar words will have similar vectors, for instance word lion and forest will have similar vectors [8,9]. We can identify both semantic and syntactic similarity using this word embedding. This method utilizes "**Cosine similarity**" to find out the closeness of words. Quoting the famous example here, we can do king - man = queen. Word embedding can be obtained using two methods [10,11,12,13]:

- CBOW model
- Skip-gram model

1.4. CBOW model and Skip-gram model

A continuous bag of words (CBOW), which determines the word based on context. Whereas Skip-gram predicts context based on word [5,6,7]. These models capture syntactic and semantic similarities between words; cosine similarity is utilized to find similar words and both models use neural networks to produce word embedding.

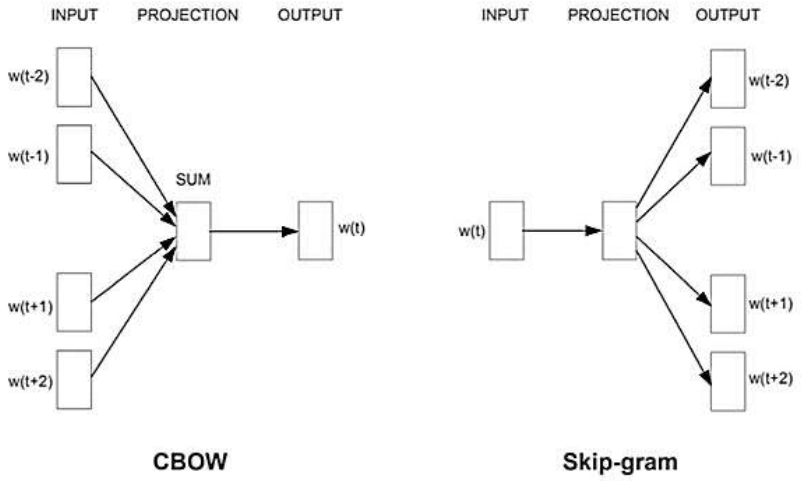

Figure 1.41. Architecture of the CBOW and skip-gram

In the given, Figure 1.41 we can see the word embedding for the word “trees”, now using this word embedding the models can predict similar words as shown in Figure 1.42.

```
array([-3.1647831e-03,  6.6352432e-04, -4.3224380e-03, -3.2030789e-03,
       -3.2577547e-03,  2.3196186e-03, -2.1934137e-04, -3.5526946e-03,
       -4.3959594e-03,  3.6182536e-03,  5.0039287e-03, -1.6665216e-03,
        1.0310774e-03,  4.8572265e-03, -3.9254529e-03,  2.6369265e-03,
        1.4008979e-03, -3.3607757e-05,  2.3548059e-04,  5.0829898e-04,
        5.0335978e-03, -2.8610714e-03,  2.9744555e-03, -1.5639713e-04,
        2.0028965e-05, -3.4813008e-03, -4.4373344e-03,  4.3749912e-03,
        2.1769165e-03, -3.4617835e-03, -3.2318819e-03,  2.5144881e-03,
       -3.7870089e-03,  1.0379617e-03,  1.4049141e-03,  4.5789471e-03,
       -2.0588420e-03,  9.4364729e-04,  2.1561191e-03, -3.5151306e-03,
        2.8693476e-03,  4.9358369e-03,  1.8326265e-03,  6.0908357e-04,
       -3.5187874e-03,  4.5093838e-03, -2.1001834e-03,  2.5853562e-05,
        4.6728244e-03,  1.0346936e-03, -2.5840461e-03, -1.3534208e-03,
        3.7168451e-03,  5.2585325e-04,  3.7305017e-03, -4.0082322e-03,
        2.4603755e-03,  5.3490091e-04,  3.9507668e-03, -4.1897088e-04,
        2.4129493e-04, -2.0739404e-03, -1.7915292e-03, -2.8138754e-03,
       -4.9105008e-03,  4.9478668e-03,  2.8109606e-03, -2.0268252e-04,
        3.1278927e-03,  4.5318729e-03,  3.2669005e-03,  4.6174284e-03,
       -2.1689201e-03, -4.2771120e-03,  7.3322898e-04,  2.8674718e-04,
        3.5741110e-03,  5.7364105e-05,  2.8834697e-03, -3.0618829e-03,
       -3.8231914e-03,  2.9396319e-03, -2.2789377e-03,  1.2001503e-03,
       -1.1800561e-03,  3.4961426e-03,  1.1842416e-03,  5.6611816e-04,
        4.2340956e-03,  6.8602688e-04,  2.1409253e-03, -9.1481837e-04,
       -3.2595231e-03, -1.6708935e-04,  3.9600485e-04,  4.8093372e-03,
        1.4168725e-03, -1.3021023e-03, -2.4729986e-03, -3.5623604e-04],
```

Figure 1.42 Word embedding array of word “trees”

2 ARCHITECTURES OF THE MODELS:

- Pre-Processing the unstructured text data is the first stage in NLP.
- A further step may be added to the process to exclude or convert emotions to ASCII digits. A model is created in NLP, that can solve our problem,
- Input layer of the CBOW model takes the one hot encoded vector of the context word(s) of size V.
- The second layer of the model or hidden layer to the model uses N Neurons and finally the output layer returns a SoftMax vector of length V [7].
- In CBOW the order of the word does not matter. We must declare a window size, it's a tuneable parameter. Let's assume we have an environment article (processed) and we declared the window size as 2, then the context words and target word will look like in Figure 2.1:

```
[(['trees', 'play', 'role', 'for'], 'critical'),
 (['play', 'critical', 'for', 'people'], 'role'),
 (['critical', 'role', 'people', 'and'], 'for'),
 (['role', 'for', 'and', 'the'], 'people'),
 (['for', 'people', 'the', 'planet'], 'and')]
```

Figure 2.1 Context and target words.

3 EXPERIMENTS AND RESULTS:

In the above figure 2.4, the words in a list represent the context words and the words outside of it represent the target word. Our first sentence is "trees play critical role for", according to our parameter the target word should be between the first 2 words and the last 2 words in a sentence and that results in "critical" being our target word in this case [8].

After the algorithm finds out the target and context words, the one hot vector of size V is passed in the first layer of the model then the second layer of the model which contains N neurons. The second layer tries to predict the target words using the one hot encoded vector of the context words, in our case we'll pass the vectors of words trees, play, role and for in the first layer of our model then the second layer will try to predict the word "critical" then the third layer will produce a SoftMax vector of size V. Finally, the algorithm will compare the predicted target word with the actual target word and then the weights of the second layer are updated using the error.

3 CONCLUSIONS:

Skip-gram works in a similar way as that of CBOW except it uses the target word to predict the context words. To make things simple, we'll use the same example as we used in the CBOW model. For "trees play critical role for" the skip-gram model will use the label word i.e., "critical" for prediction of context words. Whether to use skip-gram or CBOW depends on the problem and the dataset that you're looking at. Skip-gram is slower, but it works well with large datasets whereas CBOW is faster and is preferred for small corpus.

2. REFERENCES

[1] Y. N. Harari, '21 Lessons for the 21st Century', Signal, 2018.

[2] P. M Nadkarni, L Ohno-Machado, and W. W. Chapman, 'Natural language processing: an introduction', Journal of the American Medical Informatics Association vol.18, no. pp. 544-551,2011

[3] J. Hirschberg and M. D. Christopher, 'Advances in natural language processing', Science, vol. 349, no. 6245, 261-266 2015.

[4] S., Krishnan, J., Wang, E., Wu, M.J. Franklin, K., Goldberg, 'Active clean: interactive data cleaning for statistical modelling', Proceedings of the VLDB Endowment, vol. 9, no. 12, pp. 948– 959, 2016.

[4] P., Bojanowski, E., Grave, A., Joulin, T., Mikolov, 'Enriching word vectors with subword information', Transactions of the association for computational linguistics vol. 5, pp.135-146, 2017.

[5] D., Coppersmith, S.J., Hong, J.R. Hosking, 'Partitioning nominal attributes in decision trees', Data Mining and Knowledge Discovery, vol. 3, no.2, pp. 197–217 1999.

[6] X., Rong, Word2vec Parameter Learning Explained, 2022.

[7] T., Mikolov, K., Chen, G. Corrado, and J., Dean, "Efficient Estimation of Word Representations in Vector Space", 2022.

[8] B., Yoshua, D., Raejean, P., Vincent, and J., Christian, "A neural probabilistic language model", The Journal of Machine Learning Research, 3: pp. 1137–1155, 2003.

[9] T., Mikolov, S., Kombrink, L., Burget, J., Cernocky, and S., Khudanpur, "Extensions of recurrent neural network language model", In Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on, pp. 5528– 5531. IEEE, 2011

[10] D., Peter, Turney, "Distributional semantics beyond words: Supervised learning of analogy and paraphrase", In Transactions of the Association for Computational Linguistics (TACL), pp. 353–366, 2013.

[11] K., Grabczewski, N., Jankowski, "Transformations of symbolic data for continuous data-oriented models", In: Artificial Neural Networks and Neural Information Processing, pp. 359–366. Springer 2003.

[12] W., Kim, B.J., Choi, E.K., Hong, S.K., Kim, D. Lee, "A taxonomy of dirty data", Data mining and knowledge discovery vol.7, no. 1, pp.81–99, 2003.

[13] K., J., Berry, P., W., Mielke, P.W., Jr, H., K., Iyer, 'Factorial designs and dummy coding', Perceptual and motor skills, vol. 87, no. 3, pp. 919–927, 1998.

[14] K. Chowdhary, 'Natural language processing', Fundamentals of artificial intelligence. pp. 603-49, 2020.