

# Representations For Words, Phrases, Sentences

Dikshant  
*B.Tech ECE (4<sup>th</sup> Sem)*  
*IIIT Hyderabad*  
dikshant.gurudutt@students.iiit.ac.in  
(+91) 9034704754

February 14, 2024

## Overview of Task

Despite my best efforts and commitment, I was unable to complete the project in its entirety due to time constraints and other prior commitments. Additionally, certain aspects of the topic posed challenges.

However, I tried to address all the essential parts of the project, with the **exception of the bonus section**. The tasks I was able to complete are as follows:

### Word Similarity Scores:

- Word to its numerical representation
- Word similarity when constraints
- Word similarity when no-constraints

### Phrase And Sentence Similarity:

- Mechanism to get representations for phrases & sentences
- Phrase Similarity
- Sentence Similarity

### Paper Reading Task

# 1 Word Similarity

**Aim:** Predict similarity score of given words.

## 1.1 Approach

- Since we are not allowed to use any pre-trained model for this and have to come up with a unsupervised/ pseudo-supervised algorithm.
- So we need to first convert the words in a corpus into some mathematical format.
- And apply some sort of unsupervised learning algo on that mathematics format to calculate the score.

## 1.2 Methodologies

Main Challenge was to find appropriate way to convert word to mathematical format [1, 2].

To formulate an algo first i've tokenized corpus based on sentences and removed words like of,the,... that make no sense in themselves.

### 1.2.1 Approaches Tried

#### Approach-1

- Broke every sentence of corpus into a 3D vector. Suppose the word is "anything strange mysterious" then the vector would become  $[[1, 0, 0], [0, 1, 0], [0, 0, 1]]$ . Similarly for other words.
- Since over here words were converted into sparse arrays, it is very hard to manipulate them.

#### Approach-2 (TF-IDF [3])

- Unlike the previous approach where the word was only assigned 1. In this approach if word is given score based on it's probability distribution.
- Suppose a word comes many times in a sentence but rarely in a corpus then its score is high on comparison with other words in a sentence.
- *Weightage of a particular word =  $TF \times IDF$ .*  
(where  $TF \equiv$  term freq. &  $IDF \equiv$  Inverse doc freq.)

$$TF(t, s) = \frac{\text{No. of occurance of } t \text{ in sentence } s}{\text{Total no. of terms in sentence } s}$$

$$IDF(t) = \ln\left(\frac{\text{Total no. of sentences in corpus}}{\text{No. of docs with term } t \text{ in them}}\right)$$

- So we can create an analogy between TF and probability also  $0 \leq TF \leq 1$ . And IDF is analogous to probability density. Where TF denotes how important a word is to a sentence and IDF denotes how important word is to a corpus.
- In my txt2vec code i've implemented IDF as following to avoid the case of  $\ln(1) = 0$ .  

$$IDF(t) = \ln\left(\frac{\text{Total no. of sentences in corpus}}{\text{No. of docs with term } t \text{ in them}}\right) + 1$$

```
[{'chapter': 2.5749641330762514,
'one': 1.6722807327088482,
'boy': 2.0464217118294576},
{'lived': 2.7869603886495833,
'mr': 2.0545521962041775,
'mrs': 2.2431547822652917},
{'dursley': 2.189707232240232,
'number': 2.5749641330762514,
'four': 2.364123280161748},
{'privet': 2.5951723403483964,
'drive': 2.520791156576993,
'proud': 2.870731864743219},
```

Figure 1: Words converted into vector

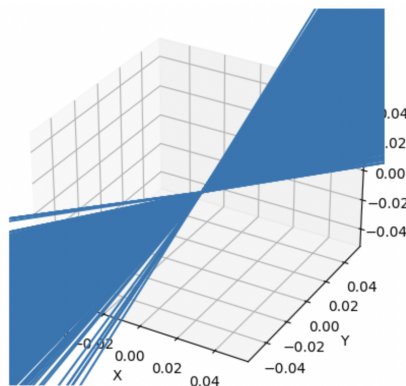


Figure 2: Word Vectors in 3d plane

## 1.3 Approach-3 (Word2Vec)

### 1.3.1 Understanding Model

- Just like any other nlp model first remove stopwords and tokenize the text.
- In my approach i've created map b/w tokens and indices and vice versa. So that conversion becomes easy.
- Since our tokens are strings so we need to encode them for encoding i'm using Approach-1 (1.2.1).
- In Word2Vec we loop through each word in sentence. in each loop we look at the words left and right of the input word. To make sense of context.
- Now consider we have 4 words in the corpus. Now to embed it we will multiply the vector representation of words through 1.2.1 with the weights matrices.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} \times \begin{pmatrix} 3 & 1 & 7 \\ 2 & 9 & 6 \\ 1 & 0 & 4 \\ 9 & 1 & 3 \end{pmatrix} = \begin{pmatrix} 3 & 1 & 7 \\ 9 & 1 & 3 \\ 1 & 0 & 4 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix}$$

$$input\ vector \times weight = embeddings$$

in embedding we are converting  $\mathbb{R}^4 \rightarrow \mathbb{R}^3$ .

- Second layer receives as input the embeddings, then from it output is generated  $\mathbb{R}^3 \rightarrow \mathbb{R}^4$ .

$$embeddings \times weight = probability\ vector$$

- after finding probability vector we need to find context prediction i.e which words are likely to be in the window of the input word.

$$softmax(probability\ vector) = prediction$$

### 1.3.2 Vector to Similarity Score

Since I've represented words as vectors and have plotted them. The easiest way to find the similarity score is to check the distance between two vectors. To simplify this I'm **using Cosine Similarity since Euclidean distances is an expensive task to run.**

- In Cosine similarity focus is on the direction of the vectors, not their magnitude.
- If two vectors are on the same side they are similar.

$$Similarity = \frac{\vec{v}_1 \cdot \vec{v}_2}{||\vec{v}_1|| \times ||\vec{v}_2||}$$

### 1.3.3 Hypothesis

- Similarity Score is always coming 1.0. This is because I'm using cosine similarity and since when I'm given two words. I'm finding those words in corpus and just using weight of that particular word. i.e if word is 'old' and 'new' then i'm getting weight of old: 2.17 and new: 2.31.
- So now when I'll find the cosine similarity it will give 1 based on the formula.

### 1.3.4 Failures

- In my code i'm not able to