Representations For Words, Phrases, Sentences

Dikshant
B.Tech ECE (4^{th} 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.
- $Weightage\ of\ a\ particular\ word\ = TF \times IDF.$
  $(where\ TF \equiv term\ freq.\ \&\ IDF \equiv Inverse\ doc\ freq.)$

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

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

- 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 $\ln(1) = 0$.
  $IDF(t) = \ln(\frac{Total\ no.\ of\ sentences\ in\ corpus}{No.\ of\ docs\ with\ term\ t\ in\ them}) + 1$

```
[{'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

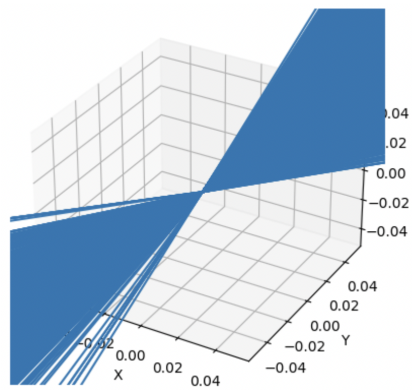

Figure 2: Word Vectors in 3d plane

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.

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

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

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

- Second layer recieves as input the embeddings, then from it outout 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.

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

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.

$$Similarity = \frac{\vec{v_1}.\vec{v_2}}{||\vec{v_1}|| \times ||\vec{v_2}||}$$

1.4 Findings

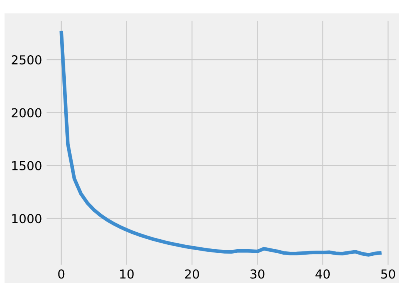

Figure 3: Error after 50 iterations

```
In [83]: v1 = get_embedding(model, "machine")
         v2 = v1 = get_embedding(model, "learning")

In [84]: similarity.fast_cosine(v1,v2)

Out[84]: 0.9996245930960868

In [85]: v1 = get_embedding(model, "machine")
         v2 = get_embedding(model, "machine")

In [86]: similarity.fast_cosine(v1,v2)

Out[86]: 1.0

In [87]: v1 = get_embedding(model, "mathematical")
         v2 = get_embedding(model, "machine")

In [88]: similarity.fast_cosine(v1,v2)

Out[88]: -0.16773641109466553

In [89]: v1 = get_embedding(model, "mathematical")
         v2 = get_embedding(model, "algorithms")

In [90]: similarity.fast_cosine(v1,v2)

Out[90]: -0.45290136337280273
```

Figure 4: Error after 50 iterations

1.5 Insights

- If we just convert words to vectors and then find the cosine similarity between them as an idea to find the similarity that would give false results. For example: if I just use tf_idf vectors and apply cosine similarity then the result would be same for each sentence.

- Also in Word2Vec approach we see that for each word it uses words before and after that word to make context. **So if some words are randomly written and we train our model on that than we would find that the output given will be very wrong.**

- Also we observe that most of the python libraries like **spaCy**, **nltk** are not able to predict the similarity score if those words were not present in the training dataset. $\implies$ **nltk and spaCy doesn't use supervised learining but rather goes with pseudo-unsupervised/ unsupervised learing for predicting similarity.**

2 Phrase and Sentence Similarity

Aim: Given labeled data-set splitted into training, testing, dev. we have to predict the Phrase and sentence similarity.

2.1 Approach

- Altough the labelled data was given. But i've used unsupervided learning in this task.
- I'm using the frequency of how many pairs have similarity in the training data as my threshold frequency. And calculating if test pairs have similarity score > threshold or not.
- Doing this would always give the similarity score to be 1. Because in the dataset[4] each pair sentences are just paraphrased.

2.2 Insights

- **for example:** consider these 2 sentences[4] to follow the approch mentioned above.
  Sentence1 - In Paris , in October 1560 , he secretly met the English ambassador , Nicolas Throckmorton , asking him for a passport to return to **England through Scotland**.
  Sentence2 - In October 1560 , he secretly met with the English ambassador , Nicolas Throckmorton , in Paris , and asked him for a passport to return to **Scotland through England** .
- In the above example all the things are same except the highlighted part. As discussed in the **Task1** that for a given word word2vec looks at words before and after to make context of the words. Due to this reason the model can't tell the difference between the two sentences.
- Now consider these examples[4]:
  Sentence1 - This was a series of nested **angular standards** , so that measurements in azimuth and elevation could be done directly in polar coordinates relative to the ecliptic .
  Sentence2 - This was a series of nested **polar scales** , so that measurements in azimuth and elevation could be performed directly in angular coordinates relative to the ecliptic .
- Over here the difference is pretty big but the model wan't able to identify the dissimilarity in these sentences. **So to tackle situations like this I also add a check to compare Parts of Speech for both Sentences**.

2.3 Findings

- To improve the accuracy of the model we need to shift from unsupervised to supervised learning.

```
actual_scores = np.array(test['label'])
correct_score_arr = np.array(correct_score)
predicted_scores = np.sum(correct_score_arr == actual_scores)
print(predicted_scores)
accuracy = predicted_scores / len(actual_scores)
print(f"Accuracy of the model is:{accuracy*100}")
```

```
4464
Accuracy of the model is:55.800000000000004
```

Figure 5: Accuracy of the model