Word Embedding:

Embedding is nothing but a representation of a textual information into a numerical. Because python model only takes numerical values are input and for that purpose we need to do word embedding.

Ex: Food is good

1 2 3 --> word embedding assigning 1,2,3 values to food, is, good

1. Good food was delivered

3 1 4 5

1. Food was not good

1 4 6 3

The entire corpus will have same number with respect to each word and number won’t change based on the records.

By using above word embedding values I can build a classifier model which cannot be upto the mark. However, when any new data comes it will have some knowledge on how to predict the output.

In the above information we have just done 1-1 mapping but we didn’t try to figure out what’s the relation between food and was/ was and food are connected. We haven’t captured whether “was” will come before “food” or after “food”

We haven’t thought of correlating these things with respect to grammar. What word will come after a noun or noun will come after verb or adjective or adverb? Have we established those relations?

These are the situation, these are the environments we haven’t captured when we are trying to do word embedding I.e., converting our data into a numerical representation.

Without considering above points, if we try to just assign numbers to words, we can build the model however the accuracy of the model will be worst. So that’s the reason there were so many people who proposed word embedding in a different different way starting from Tf-Idf, Inverse Document frequency, word to vector. With the help of state-of-the-art model we will be able to get our embeddings.

Embedding is all about generating a numerical information but if we represent the data by considering a grammatical relations, by considering the order of the word who will come after what, by considering the punctuations, by considering all the stop words, you will end up representing the data in correct format.

Models all are about relationship. If we give more correlated data which is having a best relationship it will be able to build a best model, If I’m not providing a good data which is not having a good relationship, your ai/model can’t do anything. It’s some sort of simple network that’s it.

Word to vec and doc to vector is a neural network approach and from Bert we can generate embedding from **elmo, digital Bert**.

TF-IDF: Term Frequecy Inverse Document Frequency

It’s a non-neural network approach. Tf-IDF helps us to generate word embeddings for any kind of textual information.

Term Frequency:

Many people are able to make transistion

Many people are worried about class

We are not sure about the topic

**Label Encoding**: We will group above entire data and apply “set” function to remove the duplicates. Once done create a mapping dictionary to give the numbers. Just use a map function and map all the values.

In TF-IDF, each record will be treated as a document. Hence in the above, we have 3 documents, doc1, doc2 , doc3. Now, we need to convert those 3 docs into numerical format.

Now, I will calculate term frequency and then I would like calculate Inverse Document Frequency and with the help of these 2 I will convert the entire textual data into numerical representation.

I will create a table with all unique words as columns names from my whole document. 2 row will have values and are calculated using Tf-IDF

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Many | People | Are | Able | To | Make | Transistion | Worried | About | Class | we | not | sure | a | topic |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Term Frequency: Frequency of a particular word in a particular document. Occurrence of “Many” word in doc1 is (Frequency of word occurrence / Total number of distinct words in doc1)

1. e, Tf of many in doc1 : 1/7

Inverse Document Frequency: Base 2 Logarithmic of total number of docs in whole corpus divided by occurrence of word/ frequency of word in the whole corpus

1. e., IDF of “many” in whole corpus =log( Total number of docs / Frequency of word in whole corpus)

=log( 3/2)

TF-IDF of word “many” = 1/7 X log(3/2) = 0.025

If we compare above direct numeric conversion 1,2,3 of words with Tf-IDF, we can say TF-IDF is a little bit as compared to previous numeric conversion because the reason over here is we are trying to consider atleast some sort of relationship between all the data that is given to us.

Now lets check for other word, Topic:

TF of word “Topic” in doc1 = 0/7

IDF of word “Topic” in whole corpus = log(3/1)

TF - IDF = 0/7 \*log(3/1) = 0

TF for word “are” = 1/7

IDF for word “are” = log(3/3) = log(1) = 0

TF-IDF of word “are” = 1/7 X log (1) = 0

If the word is repeating in all the docs then it will lose its relations or may be numerical value it is going to generate is equal to close to zero(0). If the word is common everywhere then it is not going to give any special or any special relationship in the sentences.

Note: The dataset which is not very much important then you will be able to find out that in terms of calculating a tf-idf, it will be able to nullify the value of particular data automatically because here we are trying to establish our relationship between our data.

When we have got the test data, we need to do the same tf-idf transformation for test data with training tf-idf model (columns from training dataset) and convert the test textual data into numeric.

When working on the test data conversion, number of docs in corpus will be sum of train and text data, hence while calculating IDF numerator will change now to (train docs +test docs) during test data conversion

If any new word present in test not in train, then we cannot input that word for model prediction as the model already trained on the train dataset without that word. Hence, we will not focus on such words during test prediction

Test data: are you able to understand - test-doc -1

Tf for “able” word = 1/5

IDF for “able” word = log2(sum of train and text docs/ frequency of word in both train and text docs)

= log2(4/2) = log2(2)

TF -IDF of word “able” in test doc 1 = 1/5\*log2(2) = 0.062

TF -IDF of word “able” in train dataset = 1/7\*log2(3) = 0.068

From above, I can say both the TF-IDF of test and train data are close to each other. If I have so many docs so much of data with all kind of words and combinations you will be able to find out that sentence and in other sentences, it will be close.