Text Similarity

Methodology:

The steps we are going to follow are:

1. Text Pre-processing
2. Feature Extraction
3. Vector Similarity

Text Pre-Processing

Data pre-processing is an essential step in building a model and depending on how well the data has been pre-processed; the results are seen.

The various text pre-processing steps that we will follow are:

1. **Tokenization**:

   Splitting the sentence into words.

2. **Lower casing:**

   Converting a word to lower case. (Dog and dog should be counted as the same)

3. **Remove punctuation and non-ASCII characters**

4. **Stop words removal:**

   Stop words are very commonly used words (a, an, the, etc.) in the documents. These words do not really signify any importance as they do not help in distinguishing two documents.

5. **Stemming or Lemmatization:**

   Stemming is a process of transforming a word to its root form.

   Unlike stemming, lemmatization reduces the words to a word existing in the language.

   We can go with either but let us go with stemming for this task as lemmatization sometimes needs 'pos' to perform more precisely.

These steps are used for dimensionality reduction.

Feature Extraction

Feature extraction is the name for methods that select and combine variables into features, effectively reducing the amount of data that must be processed, while still accurately and completely describing the original data set.

We will be using Word Embedding as our feature extraction method.

- ### Word Embedding:

Word embedding is capable of capturing context of a word in a document, semantic and syntactic similarity, relation with other words, etc.

For this, we can use either **Bert** (by Google) or **RoBerta** (by Facebook).

RoBerta is based on Bert, so it is a more accurate NLP model. The only problem is that the RoBerta model is not available as a pretrained model from TensorFlow Hub. Luckily, Bert is available, so it will be much more convenient to go ahead with Bert.

Vector Similarity

Generated word embeddings need to be compared to get semantic similarity between two vectors.

We will be using Cosine Similarity for this.

Cosine Similarity:

It is a dot product between two vectors. We would find the cosine angle between the two vectors. For degree 0, cosine is 1 and it is less than 1 for any other angle.

It is easy to implement cosine similarity from sklearn.metrics.pairwise

Conclusion

If the two texts are similar in meaning, then their cosine similarity score should be close to 1.

This cosine similarity score is the final similarity score that we want.