**About TF-IDF using the Scikit-Learn Library**

**Representing Text in Numeric Form**

Statistical algorithms use mathematics to train machine learning models. However, mathematics only work with numbers. To make statistical algorithms work with text, we first have to convert text to numbers.

For this three main approaches exist.

* Bag of Words,
* TF-IDF and
* Word2Vec.

**Bag of Words**

Bag of words scheme is the simplest way of converting text to numbers.

For instance, you have three documents:

* Doc1 = "I like to play football"
* Doc2 = "It is a good game"
* Doc3 = "I prefer football over rugby"

In the bag of words approach the first step is to create a vocabulary of all the unique words. For the above three documents, our vocabulary will be:

Vocab = [I, like, to, play, football, it, is, a, good, game, prefer, over, rugby]

**Next step**

* The next step is to convert each document into a feature vector using the vocabulary.
* The length of each feature vector is equal to the length of the vocabulary.
* The frequency of the word in the document will replace the actual word in the vocabulary.
* If a word in the vocabulary is not found in the corresponding document, the document feature vector will have zero in that place.

For instance, for Doc1, the feature vector will look like this:

Vocab = [I, like, to, play, football, it, is, a, good, game, prefer, over, rugby]

[1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]

##### TF-IDF

In the bag of words approach, each word has the same weight. The idea behind the TF-IDF approach is that the words that occur less in all the documents and more in individual document contribute more towards classification.

TF-IDF is a combination of two terms. Term frequency and Inverse Document frequency. They can be calculated as:

TF  = (Frequency of a word in the document)/(Total words in the document) IDF = Log((Total number of docs)/(Number of docs containing the word))

**TF-IDF using the Scikit-Learn Library**

##### Python's Scikit-Learn library contains the TfidfVectorizer class that can be used to convert text features into TF-IDF feature vectors.

from nltk.corpus import stopwords

from sklearn.feature\_extraction.text import

TfidfVectorizer vectorizer = TfidfVectorizer (max\_features=1500, min\_df=7, max\_df=0.8, stop\_words=stopwords.words('english'))

processed\_features = vectorizer.fit\_transform(processed\_features).toarray()

max\_features  - Most frequently occurring words which means that it only uses the 1500 most frequently occurring words to create a bag of words feature vector. Words that occur less frequently are not very useful for classification.

\*\* Note

This parameter is absolutely optional and should be calibrated according to the rational thinking and the data structure.

Sometimes it is not effective to transform the whole vocabulary, as the data may have some exceptionally rare words, which, if passed to TfidfVectorizer().fit(), will add unwanted dimensions to inputs in the future.

**Example**

from sklearn.feature\_extraction.text import TfidfVectorizer

data = ['gpu processor cpu performance',

'gpu performance ram computer',

'cpu computer ram processor jeans']

Here word jeans in the third document is hardly related and occures only once in the whole dataset. The best way to omit the word, of course, would be to use stop words parameter, but imagine if there are plenty of such words; or words that are related to the topic but occur scarcely.

In the second case, the max\_features parameter might help. If you proceed with max\_features=None, then it will create a 3x7 sparse matrix, while the best-case scenario would be 3x6 matrix:

gpu – 2 , processor = 2 , cpu = 2 , performance = 2 , ram = 2 , computer = 2 , jean=1

Total = 7 features , 3 rowa

max\_df specifies that only use those words that occur in a maximum of 80% of the documents. Words that occur in all documents are too common and are not very useful for classification.

\*\* Note

max\_df is used for removing terms that appear **too frequently**, also known as "corpus-specific stop words".

Example

* max\_df = 0.50 means "ignore terms that appear in **more than 50% of the documents**".
* max\_df = 25 means "ignore terms that appear in more than 25 documents".
* The default max\_df is 1.0, which means "ignore terms that appear in **more than 100% of the documents**". Thus, the default setting does not ignore any terms.

If max\_features is set to None, then the whole corpus is considered during the **TF-IDF** transformation. Otherwise, if you pass, say, 5 to max\_features, that would mean creating a feature matrix out of the most 5 frequent words accross text documents.

 min-df is set to 7 which shows that include words that occur in at least 7 documents.

\*\* Note

min\_df is used for removing terms that appear **too infrequently**.

Example

* min\_df = 0.01 means "ignore terms that appear in **less than 1% of the documents**".
* min\_df = 5 means "ignore terms that appear in **less than 5 documents**".

The default min\_df is 1, which means "ignore terms that appear in **less than 1 document**". Thus, the default setting does not ignore any terms.