**Feature Extraction Techniques – NLP**

This article focusses on basic feature extraction techniques in NLP to analyse the similarities between pieces of text. Natural Language Processing (NLP) is a branch of computer science and machine learning that deals with training computers to process a large amount of human (natural) language data. Briefly, NLP is the ability of computers to understand human language.

**Need of feature extraction techniques**  
Machine Learning algorithms learn from a pre-defined set of features from the training data to produce output for the test data. But the main problem in working with language processing is that machine learning algorithms cannot work on the raw text directly. So, we need some feature extraction techniques to convert text into a matrix(or vector) of features.

Some of the most popular methods of feature extraction are :

* Bag-of-Words
* TF-IDF

**Bag of Words:**  
Bag-of-Words is one of the most fundamental methods to transform tokens into a set of features. The BoW model is used in document classification, where each word is used as a feature for training the classifier.  
For example, in a task of review based sentiment analysis, the presence of words like *‘fabulous’, ‘excellent’* indicates a positive review, while words like *‘annoying’, ‘poor’* point to a negative review .  
There are 3 steps while creating a BoW model :

1. The first step is **text-preprocessing** which involves:
   1. converting the entire text into lower case characters.
   2. removing all punctuations and unnecessary symbols.
2. The second step is to **create a vocabulary** of all unique words from the corpus. Let’s suppose, we have a hotel review text.  
   Let’s consider 3 of these reviews, which are as follows :

*good movie*

*not a good movie*

*did not like*

Now, we consider all the unique words from the above set of reviews to create a vocabulary, which is going to be as follows :

1. *{good, movie, not, a, did, like}*
2. In the third step, we **create a matrix of features** by assigning a separate column for each word, while each row corresponds to a review. This process is known as **Text Vectorization**. Each entry in the matrix signifies the presence(or absence) of the word in the review. We put **1** if the word is present in the review, and **0** if it is not present.

For the above example, the matrix of features will be as follows :

| good | movie | not | a | Did | like |
| --- | --- | --- | --- | --- | --- |
| **1** | **1** | 0 | 0 | 0 | 0 |
| **1** | **1** | **1** | **1** | 0 | 0 |
| 0 | 0 | **1** | 0 | **1** | **1** |

D1: My dog is so lovely

Term/ Document Frequency

TF(My)=1/5=0.2

TF(dog)=0.2

…

Inverse Document Frequency:

= log(Total no. of documents/no of documents which particular word exist)

D1: *good movie*

*D2: not a good movie*

*D3: did not like*

A major drawback in using this model is that the order of occurence of words is lost, as we create a vector of tokens in randomised order.However, we can solve this problem by considering **N-grams**(mostly bigrams) instead of individual words(i.e. unigrams). This can preserve local ordering of words. If we consider all possible bigrams from the given reviews, the above table would look like:

| good movie | movie | did not | a | … |
| --- | --- | --- | --- | --- |
| **1** | **1** | 0 | 0 | … |
| **1** | **1** | 0 | **1** | … |
| 0 | 0 | **1** | 0 | … |

However, this table will come out to be very large, as there can be a lot of possible bigrams by considering all possible consecutive word pairs. Also, using N-grams can result in a huge **sparse**(has a lot of 0’s) matrix, if the size of the vocabulary is large, making the computation really complex!!  
Thus, we have to remove a few N-grams based on their frequency. Like, we can always remove **high-frequency N-grams**, because they appear in almost all documents. These high-frequency N-grams are generally articles, determiners, etc. most commonly called as **StopWords**.  
Similarly, we can also remove low frequency N-grams because these are really rare(i.e. generally appear in 1 or 2 reviews)!! These types of N-grams are generally typos(or typing mistakes).  
Generally, medium frequency N-grams are considered as the most ideal.  
However, there are some N-grams which are really rare in our corpus but can highlight a specific issue.  
Let’s suppose, there is a review that says – *“Wi-Fi breaks often”.*

Here, the N-gram *‘Wi-Fi breaks* can’t be too frequent, but it highlights a major problem that needs to be looked upon.

Our BoW model would not capture such N-grams since its frequency is really low. To solve this type of problem, we need another model i.e. **TF-IDF Vectorizer**, which we will study next.

**TF-IDF Vectorizer :**  
TF-IDF stands for *term frequency-inverse document frequency*. It highlights a specific issue which might not be too frequent in our corpus but holds great importance. The TF–IFD value increases proportionally to the number of times a word appears in the document and decreases with the number of documents in the corpus that contain the word. It is composed of 2 sub-parts, which are :

1. Term Frequency (TF)
2. Inverse Document Frequency (IDF)

**Term Frequency(TF) :**  
Term frequency specifies how frequently a term appears in the entire document.It can be thought of as the probability of finding a word within the document.It calculates the number of times a word  occurs in a review , with respect to the total number of words in the review .It is formulated as:

**Inverse Document Frequency (IDF) :**  
The inverse document frequency is a measure of whether a term is rare or frequent across the documents in the entire corpus. It highlights those words which occur in very few documents across the corpus, or in simple language, the words that are rare have high IDF score. IDF is a log normalised value, that is obtained by dividing the total number of documents  in the corpus by the number of documents containing the term, and taking the logarithm of the overall term.

Since the ratio inside the IDF’s log function has to be always greater than or equal to 1, so the value of IDF (and thus tf–idf) is greater than or equal to 0.When a term appears in large number of documents, the ratio inside the logarithm approaches 1, and the IDF is closer to 0.

**Term Frequency-Inverse Document Frequency(TF-IDF)**

A high TF-IDF score is obtained by a term that has a high frequency in a document, and low document frequency in the corpus. For a word that appears in almost all documents, the IDF value approaches 0, making the tf-idf also come closer to 0.TF-IDF value is high when both IDF and TF values are high i.e the word is rare in the whole document but frequent in a document.

Let’s take the same example to understand this better:

1. *good movie*
2. *not a good movie*
3. *did not like*

In this example, each sentence is a separate document.

Considering the bigram model, we calculate the TF-IDF values for each bigram :

|  | good movie | movie | did not |
| --- | --- | --- | --- |
| **good movie** | 1\*log(3/2) = 0.17 | 1\*log(3/2) = 0.17 | 0\*log(3/1) = 0 |
| **not a good movie** | 1\*log(3/2) = 0.17 | 1\*log(3/2) = 0.17 | 0\*log(3/1) = 0 |
| **did not like** | 0\*log(3/2) = 0 | 0\*log(3/2) = 0 | 1\*log(3/1) = 0.47 |

On a concluding note, we can say that though Bag-of-Words is one of the most fundamental methods in feature extraction and text vectorization, it fails to capture certain issues in the text. However, this problem is solved by TF-IDF Vectorizer, which also is a feature extraction method, that captures some of the major issues which are not too frequent in the entire corpus.

Co-occurrence Matrix

Ex. This is Rahul. Rahul is a Data Scientist.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | This | Is | Rahul | A | Data | Scientist |  |  |  |
| This | 0 | 1 | 0 | 0 | 0 | 0 |  |  |  |
| Is | 1 | 0 | 2 | 1 | 0 | 0 |  |  |  |
| Rahul | 0 | 2 | 0 |  |  |  |  |  |  |
| A |  |  |  |  |  |  |  |  |  |
| Data |  |  |  |  |  |  |  |  |  |
| Scientist |  |  |  |  |  |  |  |  |  |