PrinceKumarsharma

Term Frequency - Inverse Document Frequency

TfidfVectorizer converts a collection of raw documents to a matrix of TF-IDF features. Each document is represented as a set of words, and the number of times each word appears in the collection is used to compute its TF-IDF feature.

(OR)

TF-IDF is an abbreviation for Term Frequency Inverse Document Frequency. This is very common algorithm to transform text into a meaningful representation of numbers which is used to fit machine algorithm for prediction.

$$\text{tfidf}_{i,j} = \text{tf}_{i,j} \times \log\left(\frac{N}{\text{df}_i}\right)$$

$$\begin{aligned} \text{tf}_{i,j} &= \text{ total number of occurences of i in j} \\ \text{df}_i &= \text{ total number of documents (speeches) containing i} \\ N &= \text{ total number of documents (speeches)} \end{aligned}$$

TF-IDF is much more preferred than **Bag-Of-Words**, in which every word, is represented as **1 or 0**, every time it gets appeared in each Sentence, while, in TF-IDF, gives **weightage** to each Word separately, which in turn defines the importance of each word than others.

*Lets Understand these Terms separately as,

TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document), and, · IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as “is”, “of”, and “that”, may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

IDF(t) = log_e(Total number of documents / Number of documents with term t in it). Now Lets jump into the example part of it:

Let’s Consider these Three sentences:

1. He is a Good Boy
2. She is a Good Girl, and,
3. Both are Good Boy, and Girl, respectively.

So, after using Regular Expression, stop-words and other Functions from NLTK library, we get purified version of these three sentences, which can be shown as,

1. Good Boy
2. Good Girl, and
3. Good Boy Girl, respectively.

Now, Lets Consider TF(Term Frequency) operations,

Let’s assume a word “Good”, in sentence 1, as we know,

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

So, Number of times the word “Good” appears in Sentence 1 is, 1 Time, and the Total number of times the word “Good”, appears in all three Sentences is 3 times, so the TF(Term Frequency) value of word “Good” is, TF(“Good”)=1/3=0.333.

Now, lets consider the value of each word, with reference to each sentence, in a tabular format, which can be shown as,

So, we can see TF value of each Word with respect to each Sentence.

Now,

Lets Consider Second of TF-IDF, That is, IDF(Inverse Document Frequency) of Each word, with respect to each Sentence.

As we know,

IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

Again, lets consider, the word “Good”, in Sentence 1,Now, we know that Total Number of Sentences we have is 3(Total number of documents), also , We know the word ” Good” appears overall 3 times, considering all 3 sentences, so, Number of documents with term “Good” in it=3,

So, IDF (Inverse Document Frequency) Value of word “Good” would be “ Log(3/3)”, Now, lets consider the IDF(Inverse Document Frequency) Value of each word, in a Tabular For,

Now, We have Values for both, TF(Term Frequency) as well as IDF(Inverse Document Frequency) for each word, for each Sentence we have, So, Finally the TF-IDF Value for each word would be= TF(Value)*IDF(Value).

Let's Present TF-IDF Value of each word in a tabular form given below,

So,

As a Conclusion, we can see that, the word "Good", appears in each of these 3 sentences, as a result the Value of the word "Good" is Zero, while the Word "Boy" appears only 2 times, in each of these 3 sentences, As a results, we can see, in Sentence 1, the Value(Importance) of word "Boy" is more than the Word "Good".

As a result, we can see that, TF-IDF, gives Specific Value or Importance to each Word, in any paragraph, The terms with higher weight scores are considered to be more importance, as a result TF-IDF has replaced the concept of "Bag-Of-Words" which gives the Same Value to each word, when occurred in any sentence of Paragraph, which is Major Disadvantage of Bag-Of-Words.

TF-IDF was invented for document search and can be used to deliver results that are most relevant to what you're searching for. Imagine you have a search engine and somebody looks for "Dog". The results will be displayed in order of relevance. That's to say the most relevant Pet articles will be ranked higher because TF-IDF gives the word "Dog" a higher score.

Well, This is How TF-IDF(Term Frequncy and Inverse Document Frequency) works!

If you Like this Blog, Do leave a "Clap", and Share with your Friends, Colleagues.

Thank-you 😊.

Connect with me: email : princekr301@gmail.com (mailto:princekr301@gmail.com) My GitHub Profile : https://github.com/princekr301 (https://github.com/princekr301) My LinkedIn Profile :https://www.linkedin.com/in/prince-kumar-data-science-301pks/ (https://www.linkedin.com/in/prince-kumar-data-science-301pks/)

Import Library

In [18]:
```
import pandas as pd
import numpy as np
from nltk import word_tokenize,sent_tokenize
from nltk.stem import PorterStemmer,WordNetLemmatizer
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfTransformer,TfidfVectorizer
```

In [19]:
```
x = ["Prince data science trainer","trainer teaches data science","data science knows about prince",
     "prince working with data science"]
```

In [3]:
```
df = pd.DataFrame({"Title":x})
```

In [4]:
```
df
```

Out[4]:
```
                            Title
0          Prince data science trainer
1         trainer teaches data science
2   data science knows about prince
3    prince working with data science
```

In [5]:
```
df["Title"] = df["Title"].apply(lambda x:x.lower())
```

In [6]:
```
df
```

Out[6]:
```
                            Title
0          prince data science trainer
1         trainer teaches data science
2   data science knows about prince
3    prince working with data science
```

In [7]:
```
WordNetLemmatizer().lemmatize("training",pos="v")
```

Out[7]:
```
'train'
```

In [8]:
```
PorterStemmer().stem("training")
```

Out[8]:
```
'train'
```

In [9]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [10]:
```
tfidf = TfidfVectorizer()
```

```
In [11]: x = tfidf.fit_transform(df["Title"]).toarray()
         x

Out[11]: array([[0.        , 0.41599288, 0.        , 0.50881901, 0.41599288,
                 0.        , 0.6284927 , 0.        , 0.        ],
                [0.        , 0.35455723, 0.        , 0.        , 0.35455723,
                 0.67943473, 0.53567415, 0.        , 0.        ],
                [0.58202057, 0.30372248, 0.58202057, 0.37149619, 0.30372248,
                 0.        , 0.        , 0.        , 0.        ],
                [0.        , 0.30372248, 0.        , 0.37149619, 0.30372248,
                 0.        , 0.        , 0.58202057, 0.58202057]])
```

```
In [12]: tfidf.vocabulary_

Out[12]: {'prince': 3,
          'data': 1,
          'science': 4,
          'trainer': 6,
          'teaches': 5,
          'knows': 2,
          'about': 0,
          'working': 8,
          'with': 7}
```

```
In [13]: n = tfidf.get_feature_names_out()
```

```
In [14]: len(n)

Out[14]: 9
```

```
In [15]: n

Out[15]: array(['about', 'data', 'knows', 'prince', 'science', 'teaches',
                'trainer', 'with', 'working'], dtype=object)
```

```
In [16]: pd.DataFrame(x,columns=n)

Out[16]:
```

	about	data	knows	prince	science	teaches	trainer	with	working
0	0.000000	0.415993	0.000000	0.508819	0.415993	0.000000	0.628493	0.000000	0.000000
1	0.000000	0.354557	0.000000	0.000000	0.354557	0.679435	0.535674	0.000000	0.000000
2	0.582021	0.303722	0.582021	0.371496	0.303722	0.000000	0.000000	0.000000	0.000000
3	0.000000	0.303722	0.000000	0.371496	0.303722	0.000000	0.000000	0.582021	0.582021

```
In [17]: df

Out[17]:
```

	Title
0	prince data science trainer
1	trainer teaches data science
2	data science knows about prince
3	prince working with data science

(Term Frequency) X (Inverse Document frequency)

Term Frequency * Inverse Document frequency It is a natural language Processing techniques with the help fo this techinque we are converting the text to vector According to the word waitage.its create a sparse matrix Its ignore the out of vocabulary word it is the biggest disadvanteag of the natural language processing

insights

SO It is my complete tf*idf tutorial

TF-IDF (Term Frequency-Inverse Document Frequency) is a numerical statistic that is used to evaluate how important a word is to a document in a collection or corpus.

There is a few Advantage of this tfidf vectorizer

1. Information Retreval from this dataset
2. its show the waitage according to his importance.
3. Its show the decimal values.

There is a few Disadvantage of the TF (X) IDF

1. It's create sparse matrix it means there is high saparcity.
2. There is OOV Problem Out of Vocabulary
3. Its create high dimention of matrix
4. Does not understand the symentic meaning

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```