**TF-IDF Vectorizer scikit-learn**

1. **Introduction**
   1. **Keyword Extraction**

It is a text analysis technique that automatically extracts the most used and most important words and expressions from a text. It helps summarize the content of texts and recognize the main topics discussed.

**Example**: Imagine you want to analyze thousands of online reviews about your product. Keyword extraction helps you sift through the whole set of data and obtain the words that best describe each review in just seconds. That way, you can easily and automatically see what your customers are mentioning most often, saving your teams hours upon hours of manual processing.

* 1. **Why is Keyword Extraction Important?**

With keyword extraction you can find the most important words and phrases in massive datasets in just seconds. These words and phrases can provide valuable insights into topics your customers are talking about.

**Example**: What percentage of customer reviews are saying something related to Price? How many of them are talking about UX? These insights can help you shape a data-driven business strategy by identifying what customers consider important, the aspects of your product that need to be improved, and what customers are saying about your competition, among others.

* Help you something:
* Automatically index data
* Summarize a text
* Generate tag clouds with the most representative keywords
  1. **How does keyword extraction work?**

There are different approaches to keyword extraction:

There are different types of statistical approaches: word frequency, word collocations and co-occurrences, TF-IDF, and RAKE (Rapid Automatic Keyword Extraction).

1. **Bag of words**

Machine learning algorithms cannot work with raw text directly. Rather, the text must be converted into vectors of numbers. In natural language processing, a common technique for extracting features from text is to place all of the words that occur in the text in a **bucket**. This approach is called a **bag of words (BoW)**.

* By casting the bag of words to a set, **duplicate** **words** will be automatically **removed**.
* A dictionary of words and their occurrence for each document in the corpus (collection of documents) will be created.

**Stop words**

Useless words are referred to as **stop words**, the python natural language toolkit library provides a list of English stop words

from nltk.corpus import stopwords

stopwords.words(‘english’)

* When building a model with the goal of understanding text, all stop words will be removed.
* Another strategy is to score the relative importance of words using **TF-IDF**.

1. **TF-IDF**
   1. **What is it?**

*TF-IDF is an abbreviation for Term Frequency Inverse Document Frequency.*

***Meaning: is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. In other word, this metric calculates the number of times a word appears in a text (term frequency) and compares it with the inverse document frequency (how rare or common that word is in the entire dataset).***

* The higher the score is, the more relevant the word is to the document.
* In many of these cases, the words that appear more frequently in a group of documents are not necessarily the most relevant. Likewise, a word that appears in a single text but doesn’t appear in the remaining documents may be very important to understand the content of that text.

**Background:**

It was introduced in 1972 by Karen Spärck Jones with title “term specificity”. Instead of representing a term in a document by its raw frequency (number of occurrences) or its relative frequency, each term is weighted by dividing the term frequency by the number of documents in the corpus containing the word.

**Why**: to avoid common problem when conducting text analysis:

* The most frequently used words in a document are often the most frequently used words in all of the documents.
* In contrast, terms with the highest **tf-idf** scores are the terms in a document that are distinctively frequent in a document, when that document is compared to other documents.
  1. **Purpose**
* *For information retrieval:*

*Imagine you have a search engine and somebody looks for* ***LeBron****. The results will be displayed in order of relevance. That’s to say the most relevant sports articles will be* ***ranked higher because TF-IDF gives the word LeBron a higher score****.*

* *Keyword extraction:*

*The* ***highest scoring words*** *of a document are the* ***most relevant to that document****, and therefore they can be considered***keywords***for that document.*

* 1. **How it works**

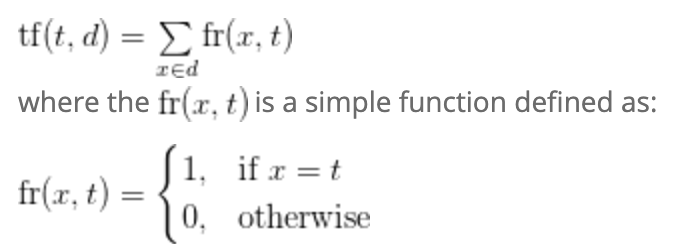
*By increasing proportionally to the number of times a word appears in a document, but is offset by the number of documents that contain the word.*

*It means: words that are common in every document, such as this, what, and if, rank low even though they may appear many times, since they don’t mean much to that document in particular.*

*However, if the word* ***Bug*** *appears many times in a document, while not appearing many times in others, it probaly means that it’s very relevant.*

*For example, if what we’re doing is trying to find out which topics some NPS responses belong to, the word* ***Bug*** *would probably end up being tied to the topic Reliability, since most reponses containing that word would be about that topic.*

* 1. **How to transform**



Where:

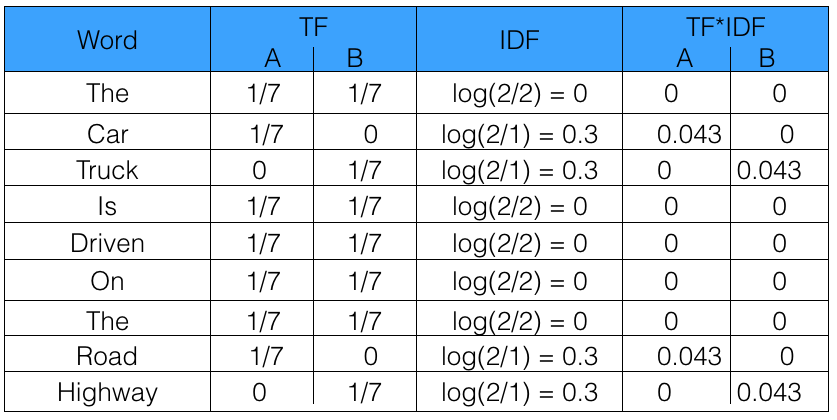
n: number of documents

: document frequency of the term t/ the number of documents where the term t appears.

* 1. **Example**

*Sentence 1: The car is driven on the road.*

*Sentence 2: The truck is driven on the highway.*



* 1. **Coding**

#first step is to import the library  
from sklearn.feature\_extraction.text import TfidfVectorizer

#for the sentence, make sure all words are lowercase or you will #run into error. for simplicity, I just made the same sentence #all lowercase

firstV= 'the car is driven on the road'  
secondV= 'the truck is driven on the highway'

#calling the TfidfVectorizer  
vectorize= TfidfVectorizer()

#fitting the model and passing our sentences right away:  
response= vectorize.fit\_transform([firstV,secondV])

print(response)

**References**

1. <https://towardsdatascience.com/tf-idf-explained-and-python-sklearn-implementation-b020c5e83275>
2. <https://medium.com/@cmukesh8688/tf-idf-vectorizer-scikit-learn-dbc0244a911a>
3. <https://bktranquangchung.wordpress.com/2018/09/10/tf-idf/>
4. <https://programminghistorian.org/en/lessons/analyzing-documents-with-tfidf#tf-idf-definition-and-background>
5. <https://monkeylearn.com/keyword-extraction/>
6. <https://programminghistorian.org/en/lessons/analyzing-documents-with-tfidf#tf-idf-definition-and-background>
7. <https://medium.com/@imamun/creating-a-tf-idf-in-python-e43f05e4d424>
8. <https://github.com/kavgan/nlp-in-practice/tree/master/tf-idf>