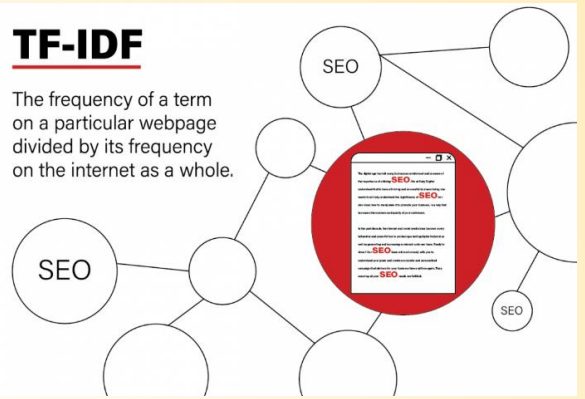
DOUGLAS COLLEGE



FUNDAMENTALS OF MACHINE LEARNING IN DATA SCIENCE

CSIS 3290

WORD FREQUENCY MATRIX WITH TFIDF VECTORIZER

FATEMEH AHMADI

TF-IDF

- **TF-IDF (term frequency-inverse document frequency)** is a statistical measure that evaluates how relevant a word is to a document in a collection of documents.
- This is done by multiplying two metrics: how many times a word appears in a document (TF), and the inverse document frequency (IDF) of the word across a set of documents.
- It has many applications such as in automated text analysis, and it is very useful for scoring words in machine learning algorithms for Natural Language Processing (NLP).
- TF-IDF was invented for document search and information retrieval. 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. So, 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.

TF-IDF

https://monkeylearn.com/blog/what-is-tf-idf/

- The **term frequency** of a word in a document. There are several ways of calculating this frequency, with the simplest being a raw count of instances a word appears in a document. Then, there are ways to adjust the frequency, by length of a document, or by the raw frequency of the most frequent word in a document.
- The **inverse document frequency** of the word across a set of documents. This means, how common or rare a word is in the entire document set. The closer it is to 0, the more common a word is. This metric can be calculated by taking the total number of documents, dividing it by the number of documents that contain a word, and calculating the logarithm.
- So, if the word is very common and appears in many documents, this number will approach 0. Otherwise, it will approach 1.

TFIDF

- **Term Frequency:**

In document d, the frequency represents the number of instances of a given word t. Therefore, we can see that it becomes more relevant when a word appears in the text, which is rational. Since the ordering of terms is not significant, we can use a vector to describe the text in the bag of term models. For each specific term in the paper, there is an entry with the value being the term frequency.

```
tf(t,d) = count of t in d / number of words in d
```

https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/

TFIDF

- **Document Frequency:**

This tests the meaning of the text, which is very similar to TF, in the whole corpus collection. The only difference is that in document **d**, **TF** is the frequency counter for a term **t**, while **df** is the number of occurrences in the document set **N** of the term **t**. In other words, the number of papers in which the word is present is **df**.

```
df(t) = occurrence of t in documents
```

https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/

TFIDF

https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/

- **Inverse Document Frequency:**

Mainly, it tests how relevant the word is. The key aim of the search is to locate the appropriate records that fit the demand. Since **tf** considers all terms equally significant, it is therefore not only possible to use the term frequencies to measure the weight of the term in the paper. First, find the document frequency of a term **t** by counting the number of documents containing the term:

```
df(t) = N(t)
where
df(t) = Document frequency of a term t
N(t) = Number of documents containing the term t
```

Term frequency is the number of instances of a term in a single document only; although the frequency of the document is the number of separate documents in which the term appears, it depends on the entire corpus. Now let's look at the definition of the frequency of the inverse paper. The IDF of the word is the number of documents in the corpus separated by the frequency of the text.

```
idf(t) = N/ df(t) = N/N(t)
```

TFIDF

The more common word is supposed to be considered less significant, but the element (most definite integers) seems too harsh. We then take the logarithm (with base 2) of the inverse frequency of the paper. So the if of the term **t** becomes:

$$idf(t) = log(N/ df(t))$$

https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/

TFIDF

- **Computation:** Tf-idf is one of the best metrics to determine how significant a term is to a text in a series or a corpus. tf-idf is a weighting system that assigns a weight to each word in a document based on its term frequency (tf) and the reciprocal document frequency (tf) (idf). The words with higher scores of weight are deemed to be more significant.

Usually, the tf-idf weight consists of two terms-

1. **Normalized Term Frequency (tf)**
2. **Inverse Document Frequency (idf)**

```
tf-idf(t, d) = tf(t, d) * idf(t)
```

Applications of TF-IDF

- **Information retrieval**

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 Java. The results will be displayed in order of relevance. That's to say the most relevant programming language articles or those describing Java island will be ranked higher because TF-IDF gives the word Java a higher score.

It's likely that every search engine you have ever encountered uses TF-IDF scores in its algorithm.

- **Keyword Extraction**

TF-IDF is also useful for extracting keywords from text. How? The highest scoring words of a document are the most relevant to that document, and therefore they can be considered *keywords* for that document.

TFIDF

In [9]:
```
import pandas as pd
from sklearn.pipeline import make_pipeline
from sklearn.cluster import KMeans
from scipy.sparse import csr_matrix
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.decomposition import TruncatedSVD
```

CSR Matrix

- In **Word Frequency Matrix**, there could be lots of zero values. As a result, such as matrix is a sparse matrix.
- **CSR matrix** (*Compressed Sparse Rows*) only stores non zero values while **numpy.array** stores all zero and non zero values. It is helpful in terms of saving memory space.
- The problem with CSR matrix is that PCA cannot be run on that. So, we use another approach as **SVD (***Singular Value Decomposition***).**

TfidfVectorizer

- **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.
- The **TfidfVectorizer** class implements a vectorizer for calculating **term frequency/inverse document frequency (TF/IDF)**, which can be used as a measure for modeling the **importance** of terms within documents. It takes in raw texts and returns an array of feature vectors for each input text.

https://www.egochi.com/tfidfvectorizer/

TfidfVectorizer and CSR Matrix

```
In [14]: docs=['First try with SVD and CSR','First with Pipeline and second with KMeans']
         title=['First doc','Second doc']

In [15]: tf1=TfidfVectorizer()

In [16]: csr1=tf1.fit_transform(docs)
```

CSR Matrix

```
In [16]: csr1=tf1.fit_transform(docs)
```

```
In [17]: print(csr1.toarray())
```

```
[[0.33471228 0.47042643 0.33471228 0.         0.         0.
  0.47042643 0.47042643 0.33471228]
 [0.2895694  0.         0.2895694  0.40697968 0.40697968 0.40697968
  0.         0.         0.57913879]]
```

```
In [18]: print(csr1)
```

```
  (0, 1)        0.4704264280854632
  (0, 0)        0.3347122780719073
  (0, 6)        0.4704264280854632
  (0, 8)        0.3347122780719073
  (0, 7)        0.4704264280854632
  (0, 2)        0.3347122780719073
  (1, 3)        0.4069796831885955
  (1, 5)        0.4069796831885955
  (1, 4)        0.4069796831885955
  (1, 0)        0.28956939652270214
  (1, 8)        0.5791387930454043
  (1, 2)        0.28956939652270214
```

TfidfVectorizer

Two methods of TfidfVectorizer

```
In [19]: words1=tf1.get_feature_names_out()

In [20]: print(words1)
         ['and' 'csr' 'first' 'kmeans' 'pipeline' 'second' 'svd' 'try' 'with']

In [21]: print(tf1.get_stop_words())
         None
```

TfidfVectorizer

sklearn.feature_extraction.text.TfidfVectorizer

class **sklearn.feature_extraction.text.TfidfVectorizer**(*, *input='content', encoding='utf-8', decode_error='strict', strip_accents=None, lowercase=True, preprocessor=None, tokenizer=None, analyzer='word', stop_words=None, token_pattern='(?u)\b\w\w+\b', ngram_range=(1, 1), max_df=1.0, min_df=1, max_features=None, vocabulary=None, binary=False, dtype=<class 'numpy.float64'>, norm='l2', use_idf=True, smooth_idf=True, sublinear_tf=False*) [source]

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

https://www.geeksforgeeks.org/using-countvectorizer-to-extracting-features-from-text/

CountVectorizer

- **CountVectorizer:** It is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text. This is helpful when we have multiple such texts, and we wish to convert each word in each text into vectors (for using in further text analysis).
- **CountVectorizer** creates a matrix in which each unique word is represented by a column of the matrix, and each text sample from the document is a row in the matrix. The value of each cell is nothing but the count of the word in that particular text sample. This can be visualized as follows:

	at	each	four	geek	geeks	geeksforgeeks	help	helps	many
document[0]	0	0	0	1	1	0	0	1	0
document[1]	0	0	1	0	2	0	1	0	0
document[2]	1	1	0	1	1	1	0	1	1

CountVectorizer

```
In [23]: vect1=CountVectorizer()

In [24]: vect1.fit(docs)

Out[24]: ▾ CountVectorizer
         CountVectorizer()

In [25]: print("Vocabulary: ", vect1.vocabulary_)

         Vocabulary:  {'first': 2, 'try': 7, 'with': 8, 'svd': 6, 'and': 0, 'csr': 1, 'pipeline': 4, 'second': 5, 'kmeans': 3}
```

```
In [27]: vector = vect1.transform(docs)

In [29]: print("Encoded Document is:")
         print(vector.toarray())

         Encoded Document is:
         [[1 1 1 0 0 0 1 1 1]
          [1 0 1 1 1 1 0 0 2]]
```

SVD

```
In [42]: docs2=['First try with SVD and CSR','First with Pipeline and second with KMeans','First with SVD','Pipeline First']
         title2=['First doc','Second doc','Third doc','Forth doc']

In [46]: csr2=tf1.fit_transform(docs2)
```

```
In [43]: svd1=TruncatedSVD(n_components=2)

In [44]: kmeans1=KMeans(n_clusters=2)
```

Singular value decomposition (**SVD**) and principal component analysis (**PCA**) are two eigenvalue methods used to reduce a high-dimensional data set into fewer dimensions while retaining important information. Also, **Truncated SVD** is a technique used for dimensionality reduction in machine learning.

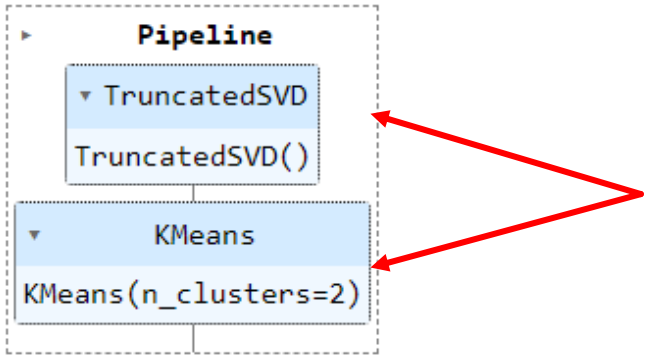
Pipeline

```
In [45]: pipeline1=make_pipeline(svd1,kmeans1)

In [47]: pipeline1.fit(csr2)
```

```
D:\Anaconda\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
D:\Anaconda\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning: KMeans is known to have a memory leak on Windows wi
th MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS
=1.
  warnings.warn(
```

Out[47]:



- The pipeline is a Python Scikit-learn utility for orchestrating machine learning operations.
- Pipelines function by allowing a linear series of data transforms to be linked together, resulting in a measurable modeling process.

https://www.askpython.com/python/examples/pipelining-in-python

Prediction

```
In [49]: labels2=pipeline1.predict(csr2)

In [50]: df=pd.DataFrame({'Labels':labels2,'Docs':docs2})

In [51]: df1=df.sort_values('Labels')
         print(df1)
```

	Labels	Docs
1	0	First with Pipeline and second with KMeans
3	0	Pipeline First
0	1	First try with SVD and CSR
2	1	First with SVD

Real labels

Predicted labels