1 **Document-Term Matrix (DTM) or Document-Word Matrix (DW):**

- Represents the relationship between documents and words.
- Each row represents a document, and each column represents a word.
- The value of each cell is the frequency of the word appearing in that document.
- Used for document classification, search, topic modeling, etc.

2 **Word-Word Matrix (WW) or Co-occurrence Matrix:**

- Represents the relationship between words.
- Each row and column represents a word.
- The value of each cell is the frequency of a specific word appearing together with another word.
- Used for word embedding, similarity measurement, association analysis, etc.

TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) Definition:

- A method of evaluating the **importance** of a word by **multiplying** the Term Frequency (TF) and the Inverse Document Frequency (IDF).

- Assigns weights to each word in the DTM to **reflect the importance of the wor**d.
- Mainly used for document similarity calculation, determining the importance of search results in search systems, and calculating the importance of specific words within a document

TF

tf(d,t) : 특정 문서 d에서의 특정 단어 t의 등장 횟수

Measures how often a word appears in a document. A higher frequency suggests greater importance. If a term appears frequently in a document, it is likely relevant to the document's content.

$$TF(t, d) = \frac{\text{Number of times term t appears in document d}}{\text{Total number of terms in document d}}$$

Limitations of TF Alone:

- TF does not account for the global importance of a term across the entire corpus.
- Common words like **"the"** or **"and"** may have high TF scores but are not meaningful in distinguishing documents.

IDF

df(t) : 특정 단어 t가 등장한 문서의 수.
idf(t) : df(t)에 반비례하는 수.

Reduces the weight of common words across multiple documents while increasing the weight of rare words. If a term appears in fewer documents, it is more likely to be meaningful and specific.

$$IDF(t, D) = \log \frac{\text{Total number of documents in corpus D}}{\text{Number of documents containing term t}}$$

Limitations of IDF Alone:

- IDF does not consider how often a term appears within a specific document.
- A term might be rare across the corpus (high IDF) but irrelevant in a specific document (low TF).

Converting text into TF - IDF

Given three documents :

1 **Document 1:** "The cat sat on the mat."
2 **Document 2:** "The dog played in the park."
3 **Document 3:** "Cats and dogs are great pets."

Q : to calculate the TF-IDF score for "Cat"

Step 1: Calculate Term Frequency (TF)

For Document 1:

- The word **"cat"** appears **1 time**.
- The total number of terms in Document 1 is **6** ("the", "cat", "sat", "on", "the", "mat").
- So, TF(cat,Document 1) = **1/6 (0.167)**

For Document 2:

- The word **"cat"** does **not appear**.
- So, TF(cat,Document 2)=0

.

For Document 3:

- The word **"cat"** appears **1 time** (as "cats").
- The total number of terms in Document 3 is **6** ("cats", "and", "dogs", "are", "great", "pets").
- So, TF(cat,Document 3)= **1/6 (0.167)**

Step 2: Calculate Inverse Document Frequency (IDF)

- **Total number of documents in the corpus (D):** 3
- **Number of documents containing the term "cat":** 2 (Document 1 and Document 3).

IDF(cat, D) = **log 3/2 =0.176**

Step 3: Calculate TF-IDF

- **Total number of documents in the corpus (D):** 3
- **Number of documents containing the term "cat":** 2 (Document 1 and Document 3).

Formula: TF-IDF(t,d,D) = TF(t,d) * IDF(t,D)

Document 1 : 0.167*0.176 =0.0293333333
Document 2 : 0 * 0.176
Document 3 : 0.167*0.176 =0.0293333333

Step by step using sklearn

```
# import required module
from sklearn.feature_extraction.text import TfidfVectorizer

# assign documents
d0 = 'Geeks for geeks'
d1 = 'Geeks'
d2 = 'r2j'

# merge documents into a single corpus
string = [d0, d1, d2]


# create object
tfidf = TfidfVectorizer()

# get tf-df values
result = tfidf.fit_transform(string)


# get idf values
print('\nidf values:')
for ele1, ele2 in zip(tfidf.get_feature_names(), tfidf.idf_):
    print(ele1, ':', ele2)


# get indexing
print('\nWord indexes:')
print(tfidf.vocabulary_)

# display tf-idf values
print('\ntf-idf value:')
print(result)

# in matrix form
print('\ntf-idf values in matrix form:')
print(result.toarray())
```

Step by step using python

```
import pandas as pd # 데이터프레임 사용을 위해
from math import log # IDF 계산을 위해

docs = [
'먹고 싶은 사과',
'먹고 싶은 바나나',
'길고 노란 바나나 바나나',
'저는 과일이 좋아요'
]
vocab = list(set(w for doc in docs for w in doc.split()))
vocab.sort()

# 총 문서의 수
N = len(docs)

def tf(t, d):
    return d.count(t)

def idf(t):
    df = 0
    for doc in docs:
        df += t in doc
    return log(N/(df+1))

def tfidf(t, d):
    return tf(t,d)* idf(t)


result = []

# 각 문서에 대해서 아래 연산을 반복
for i in range(N):
    result.append([])
    d = docs[i]
    for j in range(len(vocab)):
        t = vocab[j]
        result[-1].append(tf(t, d))

tf_ = pd.DataFrame(result, columns = vocab)


result = []
for j in range(len(vocab)):
    t = vocab[j]
    result.append(idf(t))

idf_ = pd.DataFrame(result, index=vocab, columns=["IDF"])
idf_


result = []
for i in range(N):
    result.append([])
    d = docs[i]
    for j in range(len(vocab)):
        t = vocab[j]
        result[-1].append(tfidf(t,d))

tfidf_ = pd.DataFrame(result, columns = vocab)
tfidf_
```