Jaccard Similarity :

$$\boxed{SIM_J(A,B) = \frac{n(A \cap B)}{n(A \cup B)}}$$

eg.: $A = \{1, 2, 3, 4\}$

$B = \{2, 5, 3\}$

$SIM_J(A,B) = \frac{2}{5} = 0.4$

Python implementation :

a) For strings : Split on spaces (" ")

b) Convert into sets

c) Take intersection and union

Shingling :

- n-gram shingles : collection/list of n words into consideration.

eg.: my_str = I am learning the concept of similarity.

unigram = {I, am, learning . . . . , similarity}

bigrams = {'I am', 'learning the', 'concept of', similarity}

Python implementation :

```
unigram = [a[i] for i in range(len(my_str.split()))]
bigram = [' '.join(a[i],a[j]) for i in range(len(my_str.split())-1)]
```

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

① TF (Term Frequency) : $f(q,D)/f(t,D)$

$f(q,D)$ = Frequency of term q in document D

$f(t,D)$ = Frequency of term t (term with maximum occurance) in document D.

② IDF (Inverse Document Frequency) : $\frac{N}{N(q)}$

N = Total no. of documents

N(q) = No. of document containing term 'q'.

Note: TF is different for diff. documents (may be same) SO TF·IDF value for a term would be different for different documents.

* Python Implementation:

① Get a list of documents

② Get a list of contents of documents.

③ def tf-idf (word, document):

```
    tf = document.count(word) / len(document)
    idf = np.log10(len(documents)/sum[1 for doc in docs if word in doc])
    return tf * idf
```

④ Get vectors of each document.

```
    vec_a = []
    vocab = set(a + b + c)
    vec_a.append(tf-idf(word, a) for word in vocab)
```

Similarly, get tf-idf vector of each document.

* BM25 : Best Match 25

- It is similar to TF·IDF with slight modifications.

$$BM25(D,q) = \frac{f(q,D) * (K+1)}{f(t,D) + K * \left(1 - b + b * \frac{D_{len}}{avgd_{len}}\right)} * \log_{10}\left(\frac{N - N(q) + 0.5}{N(q) + 0.5} + 1\right)$$

k and b are constants
usually $k = 1.25$
$b = 0.75$

{ Can be modified to optimize }

D_{len} = length of document

$avgd_{len}$ = Average length of a document

$$= \frac{\text{sum}(\text{len}(doc) \text{ for doc in documents})}{\text{len}(documents)}$$

The BM25 score indicates the relevance of a document to a query by considering the frequency of query terms in the document, the frequency of the terms in the corpus, and the length normalization of the document. It is a widely used ranking function in information retrievval and is known to provide better results than earlier ranking functions like TF-IDF.

SBERT :

SBERT stands for "Sentence-BERT," which is a pre-trained deep learning model for generating high-quality vector representations of sentences. It is a variation of BERT, which stands for "Bidirectional Encoder Representations from Transformers," a popular pre-trained model for natural language processing tasks.

SBERT uses a siamese neural network architecture to encode two input sentences into fixed-length vector representations, which can then be used to perform semantic similarity calculations or classification tasks. The siamese network consists of two identical neural networks that share the same weights and architecture.

The pre-training of SBERT involves training the model on large amounts of text data using various tasks such as masked language modeling and next sentence prediction. After pre-training, the model is fine-tuned on downstream tasks such as sentence classification or semantic similarity tasks, which can be achieved with a small amount of labeled data.

SBERT has been shown to achieve state-of-the-art performance on several benchmark datasets for sentence similarity and classification tasks. It has become a popular tool in natural language processing research and has many practical applications, including question answering, text classification, and chatbot development.

```python
from sentence_transformers import SentenceTransformer

model = SentenceTransformer('bert-base-nli-mean-tokens')
```

```python
sentence_embeddings = model.encode([a, b, c, d, e, f, g])
```

```python
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

# calculate similarities (will store in array)
scores = np.zeros((sentence_embeddings.shape[0], sentence_embeddings.shape[0]))
for i in range(sentence_embeddings.shape[0]):
    scores[i, :] = cosine_similarity(
        [sentence_embeddings[i]],
        sentence_embeddings
    )[0]
```