TF-IDF

```
import numpy as np
from sklearn.feature_extraction.text import
TfidfVectorizer,CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from scipy.spatial.distance import euclidean
from sklearn.metrics import jaccard_score
from sklearn.preprocessing import binarize

# Sample text
text1 = "Machine learning is a field of artificial intelligence."
text2 = "Deep learning is a branch of artificial intelligence and
machine learning."

# Convert texts to TF-IDF vectors
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform([text1, text2]).toarray()

# Cosine Similarity Calculation
cosine_sim = cosine_similarity([tfidf_matrix[0]], [tfidf_matrix[1]])
print(f"Cosine Similarity (TF-IDF): {cosine_sim[0][0]:.4f}")

# Euclidean Distance Calculation
euclidean_dist = euclidean(tfidf_matrix[0], tfidf_matrix[1])
print(f"Euclidean Distance (TF-IDF): {euclidean_dist:.4f}")

# Jaccard Similarity Calculation
def jaccard_similarity(vec1, vec2):
    vec1_bin = binarize([vec1])[0]
    vec2_bin = binarize([vec2])[0]
    return jaccard_score(vec1_bin, vec2_bin)

jaccard_sim = jaccard_similarity(tfidf_matrix[0], tfidf_matrix[1])
print(f"Jaccard Similarity (TF-IDF): {jaccard_sim:.4f}")
```

GLOVE

```
from gensim.downloader import load
from sklearn.metrics.pairwise import cosine_similarity
from scipy.spatial.distance import euclidean
from sklearn.metrics import jaccard_score
from sklearn.preprocessing import binarize

# Load GloVe Model
glove_model = load("glove-wiki-gigaword-50")

# Function to get embeddings
def get_embedding(text, model):
```

```
    words = text.lower().split()
    word_vectors = [model[word] for word in words if word in model]
    if not word_vectors:
        return np.zeros(model.vector_size)
    return np.mean(word_vectors, axis=0)

# Compute embeddings
embedding1 = get_embedding(text1, glove_model)
embedding2 = get_embedding(text2, glove_model)

# Cosine Similarity Calculation
cosine_sim = cosine_similarity([embedding1], [embedding2])
print(f"Cosine Similarity (GloVe): {cosine_sim[0][0]:.4f}")

# Euclidean Distance Calculation
euclidean_dist = euclidean(embedding1, embedding2)
print(f"Euclidean Distance (GloVe): {euclidean_dist:.4f}")

# Jaccard Similarity Calculation
def jaccard_similarity(vec1, vec2):
    vec1_bin = binarize([vec1])[0]
    vec2_bin = binarize([vec2])[0]
    return jaccard_score(vec1_bin, vec2_bin)

jaccard_sim = jaccard_similarity(embedding1, embedding2)
print(f"Jaccard Similarity (GloVe): {jaccard_sim:.4f}")
```

BERT MODEL

```
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity
from scipy.spatial.distance import euclidean
from sklearn.metrics import jaccard_score
from sklearn.preprocessing import binarize

# Load BERT Model
model = SentenceTransformer('all-MiniLM-L6-v2')

# Compute embeddings using BERT
embedding1 = model.encode(text1)
embedding2 = model.encode(text2)

# Cosine Similarity Calculation
cosine_sim = cosine_similarity([embedding1], [embedding2])
print(f"Cosine Similarity (BERT): {cosine_sim[0][0]:.4f}")

# Euclidean Distance Calculation
euclidean_dist = euclidean(embedding1, embedding2)
print(f"Euclidean Distance (BERT): {euclidean_dist:.4f}")
```

```
# Jaccard Similarity Calculation
def jaccard_similarity(vec1, vec2):
    vec1_bin = binarize([vec1])[0]
    vec2_bin = binarize([vec2])[0]
    return jaccard_score(vec1_bin, vec2_bin)

jaccard_sim = jaccard_similarity(embedding1, embedding2)
print(f"Jaccard Similarity (BERT): {jaccard_sim:.4f}")
```

SPACY

```
import spacy
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from scipy.spatial.distance import euclidean
from sklearn.metrics import jaccard_score
from sklearn.preprocessing import binarize

# Load spaCy's English language model with word vectors
nlp = spacy.load("en_core_web_sm")

# Function to get the average word vector for a text
def get_embedding(text):
    doc = nlp(text)
    return np.mean([token.vector for token in doc if
token.has_vector], axis=0)

# Compute embeddings using spaCy
embedding1 = get_embedding(text1)
embedding2 = get_embedding(text2)

# Cosine Similarity Calculation
cosine_sim = cosine_similarity([embedding1], [embedding2])
print(f"Cosine Similarity (spaCy): {cosine_sim[0][0]:.4f}")

# Euclidean Distance Calculation
euclidean_dist = euclidean(embedding1, embedding2)
print(f"Euclidean Distance (spaCy): {euclidean_dist:.4f}")

# Jaccard Similarity Calculation
def jaccard_similarity(vec1, vec2):
    vec1_bin = binarize([vec1])[0]
    vec2_bin = binarize([vec2])[0]
    return jaccard_score(vec1_bin, vec2_bin)

# Jaccard similarity requires binary vectors
jaccard_sim = jaccard_similarity(embedding1, embedding2)
print(f"Jaccard Similarity (spaCy): {jaccard_sim:.4f}")
```

INFORAMTION ON THE SIMILARITY MEASURES.

1. Cosine Similarity Cosine Similarity = cos (θ) = $A \cdot B \parallel A \parallel \parallel B \parallel$ Cosine Similarity=cos(θ)= ||A|||B|| A·B

Explanation:

Measures the cosine of the angle between two vectors. Ranges from -1 (exactly opposite) to 1 (exactly the same), where 0 indicates orthogonality. Suitable for high-dimensional text data like NLP tasks.

1. Euclidean Distance Euclidean Distance = ∑ i = 1 n ($a\ i - b\ i$) 2 Euclidean Distance= i=1 ∑ n (a i −b i) 2

Explanation:

Measures the straight-line distance between two points in a vector space. Values closer to 0 indicate greater similarity. Sensitive to magnitude, so data normalization may be required.

1. Jaccard Similarity Jaccard Similarity = | $A \cap B$ | | $A \cup B$ | Jaccard Similarity= |A∪B| |A∩B|

Explanation:

Measures the similarity between two sets by comparing their intersection over their union. Suitable for binary data or sparse data representations. Ranges from 0 (completely different) to 1 (completely identical).