step 1:Import Required Libraries

```
import nltk
import string
import itertools
import numpy as np

from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

from nltk.corpus import stopwords, wordnet
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

step 2: Dataset

```
documents = [
    "Machine learning is a field of artificial intelligence.",
    "Machine learning is a branch of AI.",
    "Artificial intelligence includes machine learning techniques.",
    "Deep learning is part of machine learning.",
    "Natural language processing deals with text data.",
    "NLP focuses on text and language understanding.",
    "Text similarity is important in plagiarism detection.",
    "Plagiarism detection uses text similarity measures.",
    "Cats are domestic animals.",
    "Dogs are loyal animals.",
    "The sky is blue today.",
    "It is raining heavily outside.",
    "Students are studying machine learning.",
    "Learners are studying artificial intelligence.",
    "This sentence is completely unrelated."
]

print("Total documents:", len(documents))
```

step 3: Text preproceing

```
import nltk, string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def preprocess(text):
    text = text.lower()
    text = text.translate(str.maketrans('', '', string.punctuation))
    tokens = word_tokenize(text)
    tokens = [w for w in tokens if w not in stop_words]
    tokens = [lemmatizer.lemmatize(w) for w in tokens]
    return tokens

print("Before preprocessing:")
print(documents[0])

print("\nAfter preprocessing:")
print(preprocess(documents[0]))
```

step 4: Feature Representation

```
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
```

```
processed_docs = [" ".join(preprocess(doc)) for doc in documents]

tfidf = TfidfVectorizer()
tfidf_matrix = tfidf.fit_transform(processed_docs)

bow = CountVectorizer(binary=True)
bow_matrix = bow.fit_transform(processed_docs).toarray()

print("TF-IDF shape:", tfidf_matrix.shape)
print("BoW shape:", bow_matrix.shape)
```

```
TF-IDF shape: (15, 43)
BoW shape: (15, 43)
```

step 5: Cosine Similarity

```
from sklearn.metrics.pairwise import cosine_similarity
import itertools

cosine_sim = cosine_similarity(tfidf_matrix)

pairs = []
for i, j in itertools.combinations(range(len(documents)), 2):
    pairs.append((i, j, cosine_sim[i][j]))

top_5_cosine = sorted(pairs, key=lambda x: x[2], reverse=True)[:5]

print("Top 5 Cosine Similarity Pairs:")
for i, j, score in top_5_cosine:
    print(f"Doc {i} & Doc {j} -> {score:.3f}")
```

```
Top 5 Cosine Similarity Pairs:
Doc 6 & Doc 7 -> 0.652
Doc 0 & Doc 2 -> 0.581
Doc 0 & Doc 13 -> 0.401
Doc 3 & Doc 12 -> 0.383
Doc 1 & Doc 3 -> 0.366
```

step 6: Jaccard Similarity

```
import numpy as np

def jaccard_similarity(v1, v2):
    intersection = np.sum(np.logical_and(v1, v2))
    union = np.sum(np.logical_or(v1, v2))
    return intersection / union if union != 0 else 0

jaccard_scores = []
for i, j in itertools.combinations(range(len(bow_matrix)), 2):
    jaccard_scores.append((i, j, jaccard_similarity(bow_matrix[i], bow_matrix[j])))

top_5_jaccard = sorted(jaccard_scores, key=lambda x: x[2], reverse=True)[:5]

print("Top 5 Jaccard Similarity Pairs:")
for i, j, score in top_5_jaccard:
    print(f"Doc {i} & Doc {j} -> {score:.3f}")
```

```
Top 5 Jaccard Similarity Pairs:
Doc 0 & Doc 2 -> 0.571
Doc 6 & Doc 7 -> 0.571
Doc 1 & Doc 3 -> 0.333
Doc 1 & Doc 12 -> 0.333
Doc 3 & Doc 12 -> 0.333
```

step 7: WordNet-based Semantic Similarity

```
from nltk.corpus import wordnet

def wordnet_similarity(s1, s2):
    t1 = preprocess(s1)
    t2 = preprocess(s2)
    scores = []

    for w1 in t1:
        for w2 in t2:
            syn1 = wordnet.synsets(w1)
            syn2 = wordnet.synsets(w2)
```

```
                if syn1 and syn2:
                    sim = syn1[0].wup_similarity(syn2[0])
                    if sim:
                        scores.append(sim)
        return np.mean(scores) if scores else 0

print("WordNet Semantic Similarity:")
for i in range(5):
    print(f"Doc {i} & Doc {i+1} -> {wordnet_similarity(documents[i], documents[i+1]):.3f}")
```

```
WordNet Semantic Similarity:
Doc 0 & Doc 1 -> 0.288
Doc 1 & Doc 2 -> 0.286
Doc 2 & Doc 3 -> 0.349
Doc 3 & Doc 4 -> 0.292
Doc 4 & Doc 5 -> 0.328
```

**step 8: comparison**

Comparison Results:

Cosine similarity was most effective in detecting near-copy texts, as TF-IDF emphasizes shared weighted terms. Jaccard similarity performed well only when documents shared many exact words, but failed for paraphrased content. WordNet-based similarity helped detect semantic equivalence, especially when synonyms were used instead of exact words. However, WordNet sometimes produced false positives for conceptually related but contextually different sentences. Jaccard produced the highest number of false negatives. Cosine similarity balanced accuracy and efficiency. Semantic similarity was computationally expensive but valuable for paraphrase detection.

**step 9 : lab report**

This lab demonstrated the importance of text similarity measures in NLP applications. Lexical similarity techniques such as Cosine and Jaccard rely on word overlap, while semantic similarity using WordNet captures meaning. Cosine similarity is widely used due to its robustness and efficiency. Jaccard similarity is limited by vocabulary dependence. WordNet-based similarity enhances paraphrase detection. Choosing the appropriate similarity measure depends on the nature of the text and the application.