Tribhuvan University

BHAKTAPUR MULTIPLE CAMPUS

Doodhpati-17, Bhaktapur

Lab Report of
Data Warehousing and Data Mining
(CSC 413)

Submitted by:
Mohit Neupane
Roll no: 30 (23248)

Submitted to:
Nabin Ghimire

4. Program to find similarity between documents

Theory

In the realm of information retrieval and text analysis, measuring the similarity between documents is crucial for tasks like document clustering, plagiarism detection, and recommendation systems. Document similarity refers to the degree to which two or more documents are alike in terms of their content, semantics, or structure. Various methods exist for computing document similarity, including cosine similarity, Jaccard similarity, and Euclidean distance.

Cosine similarity is a mathematical measure used to determine the similarity between two vectors in a multidimensional space, particularly prevalent in text mining, information retrieval, and recommendation systems. It calculates the cosine of the angle between two vectors, disregarding their magnitude and focusing solely on their orientation. Cosine similarity is especially useful in comparing documents represented as vectors in a high-dimensional space, where each dimension corresponds to a term or feature present in the document collection.

cosine similarity= $\| A \| \times \| B \| A \cdot B$

Program:

```
import string
def preprocess_text(text):
 text = text.lower()
 text = text.translate(str.maketrans('', '', string.punctuation))
 return text
def create_bow(text):
 words = text.split()
 word_count = {}
 for word in words:
    if word in word_count:
         word_count[word] += 1
    else:
        word_count[word] = 1
 return word_count
def calculate_similarity(bow1, bow2):
 all_words = set(bow1.keys()).union(set(bow2.keys()))
 dot_product = sum(bow1.get(word, 0) * bow2.get(word, 0) for word in
all_words)
 magnitude1 = sum(val ** 2 for val in bow1.values()) ** 0.5
 magnitude2 = sum(val ** 2 for val in bow2.values()) ** 0.5
```

```
 # Calculate cosine similarity
 if magnitude1 == 0 or magnitude2 == 0:
     return 0
 else:
        return dot_product / (magnitude1 * magnitude2)
def main():
    doc1 = "my name is mohit neupane."
    doc2 = "mohit neupane loves football."

    doc1 = preprocess_text(doc1)
    doc2 = preprocess_text(doc2)
    bow1 = create_bow(doc1)
    bow2 = create_bow(doc2)

    # Calculate similarity
    similarity = calculate_similarity(bow1, bow2)

    print("Document 1:", doc1)
    print("Document 2:", doc2)
    print("Similarity:", similarity)

if __name__ == "__main__":
    main()
```

Output

```
/mohit/Downloads/4 similarity.py
Document 1: my name is mohit neupane
Document 2: mohit neupane loves football
Similarity: 0.4472135954999579
PS C:\Users\mohit>
```

5. Implement Porter Stemmer Algorithm

Theory

The Porter Stemming Algorithm, also known as the Porter Stemmer, is a process for removing the commoner morphological and inflexional endings from words in English. It was presented by Martin Porter in 19802.

The main use of the Porter Stemmer is as part of a term normalization process that is usually done when setting up Information Retrieval systems1. The algorithm reduces words to their word stem, base or root form—generally a written word form. For example, words such as “Likes”, “liked”, “likely”, and “liking” will be reduced to “like” after stemming.

The algorithmic steps in the Porter Stemmer algorithm are:

1. A consonant in a word is a letter other than A, E, I, O, or U, and other than Y preceded by a consonant. So in TOY, the consonants are T and Y, and in SYZYGY they are S, Z, and G.
2. If a letter is not a consonant, it is a vowel. A consonant will be denoted by c, a vowel by v. A list ccc… of length greater than 0 will be denoted by C, and a list vvv… of length greater than 0 will be denoted by V.
3. Any word, or part of a word, therefore has one of the four forms: CVCV … C, VCVC … V, CVCV … VC, VCVC … C. These may all be represented by the single form [C]VCVC … [V] where the square brackets denote arbitrary presence of their contents.
4. Using (VCm) to denote VC repeated m times, this may again be written as [C] (VCm) [V]. m will be called the measure of any word or word part when represented in this form. The case m = 0 covers the null word.
5. The rules for removing a suffix will be given in the form (condition) S1 -> S2. This means that if a word ends with the suffix S1, and the stem before S1 satisfies the given condition, S1 is replaced by S2.

Program

```
def is_consonant(char):

    """Check if a character is a consonant."""

    return char.lower() not in 'aeiou'



def get_measure(word):
```

```
    """Calculate the measure of a word, which is the number of VC
sequences."""
    word = word.lower()
    count = 0
    index = 0
    while index < len(word):
        # Skip initial vowels
        while index < len(word) and not is_consonant(word[index]):
            index += 1
        # Check for consonant sequence
        if index < len(word):
            # Found a consonant, move to next vowel
            while index < len(word) and is_consonant(word[index]):
                index += 1
            # Completed a VC sequence
            count += 1
        # Skip the vowels to find the next consonant
        while index < len(word) and not is_consonant(word[index]):
            index += 1
    return count


def ends_double_consonant(word):
    """Check if a word ends with a double consonant."""
    if len(word) >= 2 and word[-1] == word[-2] and
is_consonant(word[-1]):
        return True
```

```
    return False


def ends_cvc(word):
    """Check if a word ends with a consonant-vowel-consonant sequence where the final consonant is not w, x, or y."""
    if len(word) >= 3 and is_consonant(word[-3]) and not is_consonant(word[-2]) and is_consonant(word[-1]) and word[-1] not in 'wxy':
        return True
    return False


def replace_suffix(word, suffix, replacement):
    """Replace the suffix of a word with a new suffix if it ends with the specified suffix."""
    if word.endswith(suffix):
        return word[:-len(suffix)] + replacement
    return word


def porter_stemmer(word):
    """Apply the Porter Stemmer algorithm to stem a word."""
    # Step 1a
    if word.endswith('sses'):
        word = word[:-2]  # Replace by 'ss'
    elif word.endswith('ies'):
        word = word[:-2]  # Replace by 'i'
    elif word.endswith('ss'):
        pass  # Do nothing
```

```
    elif word.endswith('s'):
        word = word[:-1]  # Remove 's'

    # Step 1b
    if word.endswith('eed'):
        if get_measure(word[:-3]) > 0:
            word = word[:-1]  # Replace by 'ee'
    else:
        if ((word.endswith('ed') or word.endswith('ing')) and
                any(is_consonant(c) for c in word[:-2])):
            word = word[:-2] if word.endswith('ed') else word[:-3]
            if word.endswith(('at', 'bl', 'iz')):
                word += 'e'
            elif ends_double_consonant(word) and not word[-1] in 'lsz':
                word = word[:-1]
            elif get_measure(word) == 1 and ends_cvc(word):
                word += 'e'

    return word

def main():
    """Test the Porter Stemmer with a list of words."""
    words = ["dogs", "cats","runs", "running", "farting", "flies",
"trouble", "troubling","probation","proudly", "singing", "troubled",
```

```
"dies", "horses", "university", "universal", "universally","compound",
"complex", "complexity"]


    print("{:15} {:15}".format("Word", "Stemmed"))
    print("-" * 30)
    for word in words:
        stemmed_word = porter_stemmer(word)
        print("{:15} {:15}".format(word, stemmed_word))


if __name__ == "__main__":
    main()
```

Output

```
/mohit/Downloads/poter 5.py
Word            Stemmed
------------------------------
dogs            dog
cats            cat
runs            run
running         run
farting         fart
flies           fli
trouble         trouble
troubling       trouble
probation       probation
proudly         proudly
singing         sing
troubled        trouble
dies            di
horses          horse
university      university
universal       universal
universally     universally
compound        compound
complex         complex
complexity      complexity
PS C:\Users\mohit>
```

6. Group the online news onto different categories like sports, entertainment, politics, etc

Theory

Naive Bayes classifiers are highly effective for news categorization tasks, where the aim is to automatically assign news articles to predefined categories like politics, sports, entertainment, finance, and technology. Leveraging word frequencies, Naive Bayes algorithms predict categories based on the features present in articles. Despite the assumption of feature independence, Naive Bayes classifiers perform well due to their ability to handle high-dimensional data efficiently and with low computational complexity. By analyzing word distributions across categories during training, Naive Bayes classifiers learn to distinguish between topics and accurately classify news articles, aiding efficient content organization and retrieval.

Formula for Naive Bayes Classification:

The Naive Bayes classification algorithm computes the posterior probability of a category CkCk given input features x1,x2,...,xnx1,x2,...,xn using Bayes' theorem:

$$P(Ck|x1,x2,...,xn)=P(Ck)\times P(x1|Ck)\times P(x2|Ck)\times...\times P(xn|Ck)P(x1)\times P(x2)\times...\times P(xn)P(Ck|x1,x2,...,xn)=P(x1)\times P(x2)\times...\times P(xn)P(Ck)\times P(x1|Ck)\times P(x2|Ck)\times...\times P(xn|Ck)$$

where P(Ck)P(Ck) is the prior probability of category CkCk, P(xi|Ck)P(xi|Ck) is the conditional probability of feature xixi given category CkCk, and P(xi)P(xi) is the marginal probability of feature xixi. The category with the highest posterior probability is assigned to the input document.

News Categorization:

News categorization, vital in information retrieval and content organization, automates the sorting of news articles into meaningful categories for efficient navigation and retrieval. With the surge of online news sources, automated categorization has become crucial. Using algorithms like Naive Bayes classifiers, support vector machines, neural networks, and deep learning models, news categorization systems classify articles based on content, keywords, topics, and metadata. These systems enhance user experience by facilitating access to relevant news articles, enabling informed decision-making, and fostering knowledge discovery in today's information-rich digital landscape.

Program

```
import string
```

```
news_data = [
    ("Kaka signs new contract with AC Milan", "sports"),
    ("Marvel announces new hollywood series", "entertainment"),
    ("Government announces new policies", "politics"),
    ("Serena Williams wins Wimbledon title", "sports"),
    ("Asap Rocky releases new album rumors", "entertainment"),
    ("Prime Minister discusses healthcare policy", "politics"),
    ("John Fish reaches finals in F1 Racing", "sports"),
    ("New Netflix series gains popularity", "entertainment"),
    ("Trade negotiations between countries", "politics"),
]

test_articles = [
    "Ricardo Kaka nominated for Balondor award",
    "Hollywood blockbuster tops box office charts",
    "Government unveils education reform plan"
]

def preprocess_text(text):
    text = text.lower()
    text = text.translate(str.maketrans('', '', string.punctuation))
    return text.split()

def build_vocabulary(data):
```

```
    vocabulary = set()
    for news_item, _ in data:
        words = preprocess_text(news_item)
        vocabulary.update(words)
    return vocabulary


def calculate_class_priors(data):
    class_counts = {}
    total_count = len(data)
    for _, category in data:
        if category in class_counts:
            class_counts[category] += 1
        else:
            class_counts[category] = 1
    priors = {category: count/total_count for category, count in
class_counts.items()}
    return priors


def calculate_word_probabilities(data, vocabulary):
    word_counts = {category: {word: 0 for word in vocabulary} for _,
category in data}
    class_counts = {category: 0 for _, category in data}


    for news_item, category in data:
        words = preprocess_text(news_item)
        class_counts[category] += len(words)
```

```
        for word in words:
            word_counts[category][word] += 1
    word_probs = {category: {word: (count + 1) /
(class_counts[category] + len(vocabulary))
                             for word, count in
word_counts[category].items()}
                  for category in class_counts}

    return word_probs

def naive_bayes_classifier(news_item, class_priors, word_probs):
    words = preprocess_text(news_item)
    scores = {category: class_priors[category] for category in
class_priors}
    for category in scores:
        for word in words:
            if word in word_probs[category]:
                scores[category] *= word_probs[category][word]

    predicted_category = max(scores, key=scores.get)
    return predicted_category

def main():
    vocabulary = build_vocabulary(news_data)
    class_priors = calculate_class_priors(news_data)
    word_probs = calculate_word_probabilities(news_data, vocabulary)
```

```
    print("Predictions:")

    for article in test_articles:

        predicted_category = naive_bayes_classifier(article,
class_priors, word_probs)

        print(f"Article: '{article}'")

        print(f"Predicted Category: {predicted_category}")

        print()


if __name__ == "__main__":

    main()
```

Output

```
PS C:\Users\mohit> & "C:/Program Files/Python311/python.exe" c:/Users
/mohit/Downloads/news_6.py
Predictions:
Article: 'Ricardo Kaka nominated for Balondor award'
Predicted Category: sports

Article: 'Hollywood blockbuster tops box office charts'
Predicted Category: entertainment

Article: 'Government unveils education reform plan'
Predicted Category: politics

PS C:\Users\mohit>
```

7. Build a recommender system for online music store

Theory

TF-IDF (Term Frequency-Inverse Document Frequency) is a crucial component in many recommender systems, particularly in the field of information retrieval and text analysis. Its computation involves two main components: term frequency (TF) and inverse document frequency (IDF). The TF of a term within a document is calculated by dividing the frequency of the term in the document by the total number of terms in the document. The IDF of a term measures its importance in the entire corpus by taking the logarithm of the ratio of the total number of documents to the number of documents containing the term. Mathematically, TF-IDF is computed by multiplying the TF of a term by its IDF. In recommender systems, TF-IDF is used to represent items or documents and to calculate the similarity between items or between items and user profiles. By considering the TF-IDF vectors of documents or items, recommender systems can effectively suggest relevant items to users based on their preferences and historical interactions, improving the overall user experience and satisfaction.

Cosine similarity, another fundamental concept in recommender systems, measures the similarity between two vectors by computing the cosine of the angle between them. In the context of recommendation algorithms, cosine similarity is often used to calculate the similarity between user profiles and item profiles represented as vectors in a high-dimensional space. By leveraging TF-IDF representations or other vector-based representations of items and users, cosine similarity helps identify items that are most similar to a user's preferences or historical interactions. Recommender systems then use cosine similarity scores to rank items and recommend the most relevant ones to users. Cosine similarity is particularly advantageous because it is unaffected by the magnitude of the vectors, focusing solely on the direction of similarity between items or user profiles. This makes it a powerful tool for providing personalized recommendations in various domains, including e-commerce, content streaming platforms, and online news portals. By incorporating TF-IDF and cosine similarity into recommendation algorithms, systems can deliver more accurate and relevant recommendations to users, enhancing user engagement and satisfaction while driving business performance.

Program

```
import pandas as pd

from sklearn.metrics.pairwise import cosine_similarity



# Sample music data (user-item matrix)

music_data = {
```

```
    'User': ['User1', 'User2', 'User3', 'User4', 'User5'],
    'Pop': [5, 8, 0, 0, 3],
    'Classical': [0, 5, 3, 1, 1],
    'Rock': [0, 1, 5, 4, 0],
    'Hip-Hop': [2, 0, 2, 0, 5],
    'Country': [0, 0,0, 5, 0],
}

# Convert data to DataFrame
df = pd.DataFrame(music_data)

# Calculate similarity matrix using cosine similarity
similarity_matrix = cosine_similarity(df.drop('User', axis=1))

# Function to recommend tracks for a given user
def recommend_tracks(user_id, similarity_matrix,
num_recommendations=3):
    user_index = df[df['User'] == user_id].index[0]
    user_similarities = similarity_matrix[user_index]
    similar_users_indices =
user_similarities.argsort()[-num_recommendations-1:-1][::-1]  # Exclude
user's own index
    recommended_tracks = []
    for index in similar_users_indices:
        similar_user_id = df.iloc[index]['User']
        user_tracks = df.iloc[index][1:]
```

```
        recommended_tracks.extend(user_tracks[user_tracks > 0].index)
# Only recommend tracks with rating > 0

    return list(set(recommended_tracks))[:num_recommendations]  #
Remove duplicates and limit recommendations




user_id1 = 'User1'

recommended_tracks = recommend_tracks(user_id1, similarity_matrix)

print("These are the recommended tracks for", user_id1, ":",
recommended_tracks)
```

Output

```
PS C:\Users\mohit> & "C:/Program Files/Python311/python.exe" c:/Users
/mohit/Downloads/musixc_7.py
These are the recommended tracks for User2 : ['Classical', 'Rock', 'P
op']
PS C:\Users\mohit> & "C:/Program Files/Python311/python.exe" c:/Users
/mohit/Downloads/musixc_7.py
These are the recommended tracks for User1 : ['Rock', 'Hip-Hop', 'Cla
ssical']
PS C:\Users\mohit>
```