Similarity Measures in Recommender Systems

Estimated Time: 15 minutes

Objectives

After this reading, you will be able to:

- Explain the significance of similarity measures in recommender systems
- Describe the different types of similarity measures

The significance of similarity measures in recommender systems

Similarity measures in recommender systems quantify the similarity between items or users based on various characteristics or behaviors. These measures are essential for tailoring recommendations to users' preferences and behaviors, forming the foundation for delivering personalized suggestions that align with users' interests.

In the context of article engagement, similarity measures play a crucial role in both **content-based and collaborative filtering recommendation systems**.

Content-based recommendations: These systems analyze the content of articles to recommend similar ones to users based on their reading history and preferences. Similarity measures help identify articles with similar topics or keywords, ensuring users receive suggestions aligned with their interests.

For example, if a user often reads articles on topics such as **deep learning** and **machine learning,** appropriate similarity measures will recommend articles that closely align with these subjects to the user.

Collaborative filtering recommendations: By analyzing user interactions with articles, collaborative filtering systems identify users with similar engagement patterns and recommend popular articles. Similarity measures enable the system to gauge the similarity between users' behavior, facilitating the delivery of relevant and engaging recommendations.

For example, say a user frequently reads articles about **data science** and **machine learning**. Meanwhile, another user has a similar reading pattern, showing interest in similar topics. Similarity measures will then help us identify the more similar users.

The need for similarity measures arises from the desire to enhance the user experience on article-reading platforms.

By leveraging these measures, platforms can:

- **Personalize recommendations:** Tailor suggestions to match users' preferences and behaviors.
- **Enhance user engagement:** Keep users interested by recommending articles they will likely enjoy.
- **Introduce diversity:** Balance personalized recommendations with introducing new and diverse content to enrich users' reading experiences.
- **Optimize recommendations:** Provide a quantitative framework for evaluating the relevance of articles, ensuring that recommendations are both pertinent and diverse.

Different types of similarity measures

The different types of similarity measures are:

- **Cosine similarity**
- **Jaccard similarity**
- **Euclidean distance**

Let us now apply similarity measures for the following real-time problem:

> **InfoSphere News Hub is a premier news aggregation service offering many articles across various domains, including politics, technology, sports, and entertainment. Within this platform, each user possesses distinct preferences and interests. The challenge at hand for InfoSphere News Hub is to curate personalized article recommendations that resonate with individual user preferences, ensuring a tailored and engaging reading experience for every user.**

Sample data representation:

Documents (Articles):

- **Document 1:** Political unrest leads to protests
- **Document 2:** New technology breakthrough announced
- **Document 3:** Team wins championship in a thrilling sports event
- **Document 4:** Popular actor's new movie release
- **Document 5:** Stock market experiences sharp rise

User profiles:

- **User 1:** Interested in politics
- **User 2:** Interested in technology and sports
- **User 3:** Interested in entertainment and finance

Cosine similarity

Cosine similarity is a metric for similarity employed to compute the cosine of the angle between two vectors, which their magnitudes then normalize.
It measures the similarity in direction between two vectors rather than their absolute values.

`Cosine similarity (A, B) = (A · B) / (||A|| * ||B||)` where A and B are two vectors.

A cosine similarity of 1 shows that the vectors are pointing in the same direction, while a cosine similarity of -1 indicates they are pointing in opposite directions.

In text analysis, cosine similarity can compare documents. Each document is a vector of word frequencies. Cosine similarity measures the directions of these vectors, indicating how similar the documents are in terms of their content.

Now let us consider the Bag of Words (BoW) representation for the documents and user profiles of the InfoSphere News Hub use case:

In the Bag of Words (BoW) representation, each document and user profile is a vector, where each vector element corresponds to a unique term or word in the vocabulary.

For example:

Document 1 (Politics) is represented by the vector $[1, 0, 0, 0, 0]$, indicating that it contains the term "Politics" and does not contain any other terms.

Document 2 (Technology) is represented by the vector $[0, 1, 0, 0, 0]$, indicating that it contains the term "Technology" and does not contain any other terms.

Document 3 (Sports) is represented by the vector $[0, 0, 1, 0, 0]$, indicating that it contains the term "Sports" and does not contain any other terms.

Document 4 (Entertainment) is represented by the vector $[0, 0, 0, 1, 0]$, indicating that it contains the term "Entertainment" and does not contain any other terms.

Document 5 (Finance) is represented by the vector $[0, 0, 0, 0, 1]$, indicating that it contains the term "Finance" and does not contain any other terms.

User profiles are similar.

User 1 (Interested in Politics) is represented by the vector $[1, 0, 0, 0, 0]$, indicating that they are interested in "Politics" and not interested in any other topics.

User 2 (Interested in Technology and Sports) is denoted by the vector $[0, 1, 1, 0, 0]$, signifying their interest in these two topics while having no interest in others.

User 3 (Interested in Entertainment and Finance) is depicted by the vector $[0, 0, 0, 1, 1]$, indicating that they are interested in "Entertainment" and "Finance" but not interested in other topics.

Mathematical calculation of cosine similarity

Here, we will be calculating the cosine similarity between 2 vectors, the `Document vector` and `User profile vector`

```
Document 1 (Politics): [1, 0, 0, 0, 0]
User 1 (Interested in Politics): [1, 0, 0, 0, 0]

Cosine Similarity(User 1, Document 1) = (1∗1 + 0∗0 + 0∗0 + 0∗0 + 0∗0) / (√1 ∗ √1) = 1/1 = 1
```

Similarly, similarity cosine similarity for User 2, Document 1 can be calculated as

```
Cosine Similarity(User 2, Document 2) = (0∗0 + 1∗1 + 1∗0 + 0∗0 + 0∗0) / (√2 ∗ √1) = 1/√2 ≈ 0.707
```

Recommendations:
Based on the cosine similarity scores, we can recommend articles to users:

User 1: Document 1 (Politics)
User 2: Document 2 (Technology) and Document 3 (Sports)
User 3: Document 4 (Entertainment) and Document 5 (Finance)

Implementation using Python

```
 1. 1
 2. 2
 3. 3
 4. 4
 5. 5
 6. 6
 7. 7
 8. 8
 9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
20. 20
21. 21
22. 22
23. 23
24. 24
25. 25
26. 26
27. 27
28. 28
29. 29
30. 30
31. 31
32. 32
33. 33
34. 34
35. 35
36. 36
37. 37
38. 38
39. 39
40. 40
41. 41
42. 42
43. 43
44. 44
45. 45
```

```
46. 46
47. 47
48. 48
49. 49
50. 50
51. 51
52. 52
53. 53
54. 54
55. 55
56. 56
57. 57
58. 58
59. 59
60. 60
61. 61
62. 62
63. 63
64. 64
65. 65
```

```
 1. Import numpy as np
 2. and import pandas as pd
 3. from sklearn. metrics.pairwise import cosine_similarity
 4.
 5. # Define the documents and their contents
 6. documents = {
 7.     ‘Document 1’: “Political unrest leads to protests.”
 8.     ‘Document 2’: “New technology breakthrough announced.”
 9.     ‘Document 3’: “Team wins championship in a thrilling sports event.”
10.     ‘Document 4’: “Popular actor’s new movie release.”
11.     ‘Document 5’: “Stock market experiences sharp rise.”
12. }
13.
14. # Create vocabulary from the documents
15. vocabulary = set()
16. for doc_content in documents.values():
17.     vocabulary.update(doc_content.lower().split())
18.
19. # Create BoW vectors for each document
20. bow_vectors = []
21. for doc_content in documents.values():
22.     bow_vector = [doc_content.lower().count(word) for word in vocabulary]
23.     bow_vectors.append(bow_vector)
24.
25. print("Document Bow vectors are", bow_vectors)
26.
27. # Convert BoW vectors to DataFrame
28. bow_df = pd.DataFrame(bow_vectors, columns=vocabulary, index=documents.keys())
29.
30. print("Document Bow Dataframe",bow_df)
31.
32. # Define user interests
33. user_interests = {
34.     'User 1': {'politics'},
35.     'User 2': {'technology', 'sports'},
36.     ‘User 3’: {‘entertainment’, ‘finance’}
37. }
38.
39. # Create user profiles as BoW vectors
40. user_profiles = {}
41. for the user, interests in user_interests.items():
42.     user_profile = [1 if word in interests else 0 for word in vocabulary]
43.     user_profiles[user] = user_profile
44.
45. print("User profile bow vectors",user_profiles)
46.
47. # Convert user profiles to DataFrame
48. user_profiles_df = pd.DataFrame(user_profiles, index=vocabulary).T
49.
50. print("user_profiles Dataframe",user_profiles_df)
51.
52. # Calculate cosine similarity between user profiles and documents
53. similarities = cosine_similarity(user_profiles_df.values, bow_df.values)
54.
55. # Create DataFrame for similarity scores
56. similarity_df = pd.DataFrame(similarities, index=user_profiles_df.index, columns=bow_df.index)
57.
58. # Recommend articles based on highest similarity scores
59. recommendations = {}
60. for user, row in similarity_df.iterrows():
61.     recommendations[user] = similarity_df.columns[row.argmax()]
62.
63. print("Recommendations:")
64. for user, article in recommendations.items():
65.     print(f"{user}: {article}")
```

Copied!

The approach taken in the Python code provided above can be summarized as follows:

- Import the necessary libraries
- Create a document dictionary called `documents` containing document names as keys and their corresponding contents as values.
- Create a `vocabulary` set that stores unique words found in all document contents by iterating through each document content, splitting it into words (using split()), converting them to lowercase, and adding them to the vocabulary set.
- Create a list of `bow_vectors`to store Bag of Words vectors for each document.
  This part iterates through each document's content and counts the occurrences of each word in the vocabulary. It creates a BoW vector for each document, where each element represents the count of a word in the vocabulary.

- Convert BoW Vectors to a DataFrame `bow_df` that stores the BoW vectors for each document. Document names are row indices, and vocabulary words are column names.
- Create a user profile dictionary `user_interests` containing user names as keys and sets of their interests as values. Each user's interests are represented as words (topics).
- Next, the `user_profile` dictionary is created to store BoW vectors representing user interests. This part iterates through each user's interests and creates a BoW vector for each user. If a term from the vocabulary matches the user's interests, it is denoted by 1; otherwise, it's represented by 0.
- User profiles dataframe `user_profiles_df` is created with user names as row indices and vocabulary words as column names.
- We use a cosine_similarity function to compute the cosine similarity matrix between the BoW vectors of user profiles and documents.
- A Pandas dataframe `similarity_df` will be created with user names as row indices and document names as column names.
- A dictionary called `recommendations` is created to store recommended articles for each user. This part iterates through each row of the similarity DataFrame, finds the document with the highest similarity score for each user, and stores it in the recommendations dictionary.
- Finally, the recommended articles for each user are printed and displayed.

Jaccard similarity

The Jaccard index, alternatively referred to as the Jaccard similarity coefficient, serves as a metric for quantifying the similarity between two sets.

It is the cardinality of the intersection of the sets divided by the cardinality of the union of the sets.

In mathematical terms, the Jaccard index for two sets, A and B, is determined as follows:

```
J(A, B) = |A ∩ B| / |A ∪ B|
```

where
`|A ∩ B|` represents the size of the intersection of sets A and B (i.e., the number of elements common to both sets).
`|A U B|`represents the size of the union of sets A and B (i.e., the total number of unique elements in both sets).

The Jaccard index spans from 0 to 1, where:

A value of 0 signifies no intersection between the sets (i.e., entirely dissimilar).
A value of 1 signifies a complete intersection between the sets (i.e., identical).

The Jaccard index is commonly used in various fields, including information retrieval, data mining, and natural language processing, to quantify the similarity between data sets or objects. It is beneficial when dealing with binary data or categorical variables.

For instance, let us compare the similarities between different documents from InfoSphereNews Hub using Jaccard similarity.

To do this, first, each document is represented as a set of words:

Document 1 (Politics): {political, unrest, leads, to, protests}
Document 2 (Technology): {new, technology, breakthrough, announced}
Document 3 (Sports): {team, wins, championship, in, thrilling, sports, event}
Document 4 (Entertainment): {popular, actor's, new, movie, release}
Document 5 (Finance): {stock, market, experiences, sharp, rise}

```
J(Document 1, Document 2)= {political, unrest, leads, to, protests}∪{new, technology, breakthrough, announced}|/|{political, unrest, leads, to, protests}∩{new, technology, breakthrough, announced}|
```

The intersection is empty, as the sets have no common words. Therefore:

J(Document 1, Document 2)=0/9=0

Implementation using python

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
```

```
import numpy as np
from sklearn.metrics import jaccard_score

# Define the Bag of Words (BoW) vectors of documents
bow_vectors = {
```

```
    'Document 1': [1, 1, 1, 0, 0, 0, 0, 0],  # BoW vector for Document 1 (Politics)
    'Document 2': [0, 1, 0, 1, 1, 0, 0, 0],  # BoW vector for Document 2 (Technology)
    'Document 3': [0, 0, 1, 1, 0, 1, 0, 0],  # BoW vector for Document 3 (Sports)
    'Document 4': [0, 0, 0, 0, 0, 1, 1, 1],  # BoW vector for Document 4 (Entertainment)
    'Document 5': [0, 0, 0, 0, 1, 0, 1, 1],  # BoW vector for Document 5 (Finance)
}

# Convert BoW vectors to binary arrays
binary_arrays = np.array(list(bow_vectors.values()))

# Calculate Jaccard similarity between pairs of documents using built-in function
jaccard_scores = {}
for i, (doc1, bow1) in enumerate(bow_vectors.items()):
    for j, (doc2, bow2) in enumerate(bow_vectors.items()):
        if i != j:  # Exclude comparing a document with itself
            jaccard_scores[(doc1, doc2)] = jaccard_score(bow1, bow2)

# Print Jaccard similarity scores
print("Jaccard Similarity Scores:")
for pair, score in jaccard_scores.items():
    print(f"{pair}: {score}")

```

Copied!

The approach taken in the Python code provided above can be summarized as follows:

- Import the necessary libraries
- Define the Bag of Words (BoW) vectors directly as lists of integers indicating the presence (1) or absence (0) of each word in the document.
- Convert the BoW vectors to a binary array using NumPy.
- Calculate the Jaccard similarity between pairs of binary arrays representing the BoW vectors using the `jaccard_score` function from scikit-learn.
- Print the Jaccard similarity scores for each pair of documents.

Euclidean distance

Euclidean distance is a metric commonly used to measure the similarity or dissimilarity between two points in an Euclidean space. In the context of recommender system similarity measures, Euclidean distance quantifies the distance between vectors representing items or users.

The Euclidean distance between two vectors in n-dimensional space is determined by taking the square root of the sum of the squared differences of corresponding elements.

`Euclidean Distance(A, B) = √(Σ((A_i - B_i)²))`
where `A_i` and `B_i` are the ith elements of vector A and B respectively.

To use Euclidean distance for similarity measurement:

Calculate the Euclidean distance between two vectors representing items or users.
The closer the distance, the greater the similarity between the vectors. Hence, considering the reciprocal of the Euclidean distance might be preferable as a measure of similarity.

Mathematical calculation of Euclidean distance

Consider the following BOW vectors

A=[1,1,1,0,0,0,0,0]
B=[0,1,0,1,1,0,0,0]

Compute the element-wise squared differences between corresponding elements of A and B.

Squared Differences = (A - B)² = [(1 - 0)², (1 - 1)², (1 - 0)², (0 - 1)², (0 - 1)², (0 - 0)², (0 - 0)², (0 - 0)²] = [1, 0, 1, 1, 1, 0, 0, 0]

Take the square root of the sum:

Euclidean Distance(A, B) = √(Sum of Squared Differences) = √(4) = 2

This shows the Euclidean distance calculation between vectors A and B, where the sum of squared differences equals 4, resulting in an Euclidean distance of 2.

A smaller Euclidean distance indicates that the objects are more similar.

The interpretation of Euclidean distance as a similarity measure may depend on the specific context and threshold chosen. For instance, a threshold could classify vectors with Euclidean distance less than or equal to a certain value as **similar** and those above it as **dissimilar**.

A Euclidean distance of 2 implies a moderate level of similarity between vectors A and B, suggesting that they share some common characteristics or features while exhibiting some differences.

Implementation using python

```
1
2
3
4
5
6
7
8
9
10
11
12
13
```

```
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
20. 20
21. 21
22. 22
23. 23
24. 24
25. 25
26. 26
```

```
 1. import numpy as np
 2.
 3. # Define the Bag of Words (BoW) vectors of documents
 4. bow_vectors = {
 5.     'Document 1': [1, 1, 1, 0, 0, 0, 0, 0],  # BoW vector for Document 1 (Politics)
 6.     'Document 2': [0, 1, 0, 1, 1, 0, 0, 0],  # BoW vector for Document 2 (Technology)
 7.     'Document 3': [0, 0, 1, 1, 0, 1, 0, 0],  # BoW vector for Document 3 (Sports)
 8.     'Document 4': [0, 0, 0, 0, 0, 1, 1, 1],  # BoW vector for Document 4 (Entertainment)
 9.     'Document 5': [0, 0, 0, 0, 1, 0, 1, 1],  # BoW vector for Document 5 (Finance)
10. }
11.
12. # Convert BoW vectors to numpy arrays
13. bow_arrays = np.array(list(bow_vectors.values()))
14.
15. # Calculate Euclidean distance between pairs of documents
16. euclidean_distances = {}
17. for i, (doc1, bow1) in enumerate(bow_vectors.items()):
18.     for j, (doc2, bow2) in enumerate(bow_vectors.items()):
19.         if i != j:  # Exclude comparing a document with itself
20.             euclidean_distances[(doc1, doc2)] = np.linalg.norm(bow1 - bow2)
21.
22. # Print Euclidean distances
23. print("Euclidean Distances:")
24. for pair, distance in euclidean_distances.items():
25.     print(f"{pair}: {distance}")
26.
```

Copied!

The approach taken in the Python code provided above can be summarized as follows:

- Import the necessary libraries
- Define each document's Bag of Words (BoW) vectors as the `bow_vectors` dictionary. Each document is represented by a list of integers, where each element corresponds to the frequency of a particular word or feature in the document as lists of integers indicating the presence (1) or absence (0) of each word in the document.
- The BoW vectors are converted into a single numpy array `bow_arrays` for easy calculation.
- Nested loops are utilized to compute the Euclidean distance between each pair of documents. For each pair of documents, the Euclidean distance is computed as the norm (length) of the difference between their respective BoW vectors, using `np.linalg.norm()` function.
- The `np.linalg.norm` function in NumPy calculates the Euclidean norm of a vector, which is the square root of the sum of the squares of its component values. This operation effectively computes the Euclidean distance between the two vectors.
- Finally, the Euclidean distances between all pairs of documents are printed out.
- The distances are stored in a dictionary `euclidean_distances` where the keys are tuples representing document pairs and the values are the corresponding Euclidean distances.

Appropriate applications of similarity measures

In recommender systems, Cosine similarity, Jaccard score, and Euclidean distance are widely used distance metrics suited to specific scenarios.

Cosine similarity
It is used when the direction of vectors matters more than their magnitudes.

Example: Recommending similar articles based on their textual content. Cosine similarity measures the angle between article vectors, effectively capturing their thematic similarity regardless of word frequency.

Jaccard score
It is effective for scenarios emphasizing set similarity, particularly with binary or categorical data.

Example: Collaborative filtering recommendation systems. Jaccard score measures user similarity based on item interactions, considering the presence or absence of interactions rather than their frequencies.

Euclidean distance

It is suitable for scenarios with crucial absolute differences between feature values.

Example: Hybrid recommendation systems combining content-based and collaborative filtering approaches. Euclidean distance can quantify similarity between users or items based on a mix of numerical and categorical features.

Each metric offers unique benefits tailored to specific recommendation scenarios, ensuring efficient and effective recommendations while accommodating diverse data types and requirements.

Conclusion

In this reading, we comprehensively understood the similarity measures and their significance in recommender systems. We demonstrated their relevance through a practical use case and provided Python code to illustrate the implementation.

Author(s)

Lakshmi Holla

Other Contributors

Malika Singla