TP: Information Retrieval Models

(Term Document Matrix and Vector Space Model)

**Problem Set:**

Learn how to create and interpret a Term Document Matrix (TDM) for a set of documents. And apply the

Vector Space Model to calculate document similarity using cosine similarity.

### **Problem 1:** Create a Term Document Matrix

**Task**: Create a Term Document Matrix for a small set of documents.

**Instructions**:

1. Choose the following three sample documents:
   * Document 1: " Data science combines statistics, computer science, and domain knowledge."
   * Document 2: " Machine learning algorithms can analyze large datasets and make predictions."
   * Document 3: " Data visualization helps in interpreting complex data and communicating insights."
2. Write a function to tokenize each document (split into words) and count the frequency of each term.
3. A screenshot of a computer

   Description automatically generatedConstruct the Term Document Matrix (TDM) and print it.

**Problem 2:** Visualize the Term Document Matrix

**Task**: Display the Term Document Matrix in a readable format.

**Instructions**:

1. Using the TDM created in Problem 1, format the matrix into a table.
2. Ensure that the rows represent documents and the columns represent terms.
3. Print the TDM with appropriate labels for documents and terms.

A screenshot of a computer

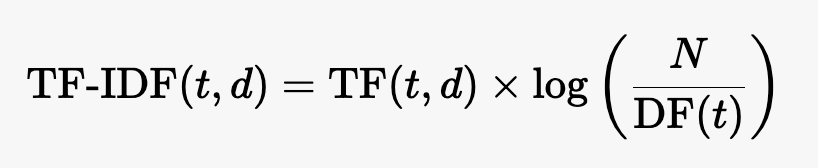
Description automatically generated

**Problem 3:** Implement TF-IDF

**Task**: Calculate TF-IDF weights for the terms in the TDM.

**Instructions**:

1. Using the TDM from Problem 1, write a function to calculate the TF-IDF for each term in each document.



1. Display the TF-IDF matrix.

import pandas as pd

from sklearn**.**feature\_extraction**.**text import TfidfVectorizer

# Define the documents

documents **=** [

**"**Data science combines statistics, computer science, and domain knowledge.**",**

**"**Machine learning algorithms can analyze large datasets and make predictions.**",**

**"**Data visualization helps in interpreting complex data and communicating insights.**"**

]

# Function to calculate TF-IDF for each term in each document

def calculate\_tfidf(documents)**:**

    # Initialize TfidfVectorizer

    vectorizer **=** TfidfVectorizer()

    tfidf\_matrix **=** vectorizer**.**fit\_transform(documents)

    # Convert the TF-IDF matrix to a DataFrame for readability

    tfidf\_df **=** pd**.**DataFrame(tfidf\_matrix**.**toarray()**,** columns**=**vectorizer**.**get\_feature\_names\_out())

    # Set the document labels

    tfidf\_df**.**index **=** [f"Document {i**+**1}" for i in range(len(documents))]

    return tfidf\_df

# Calculate and display the TF-IDF matrix

tfidf\_matrix **=** calculate\_tfidf(documents)

print(**"**TF-IDF Matrix:**"**)

tfidf\_matrix

TF-IDF Matrix:

|  | **algorithms** | **analyze** | **and** | **can** | **combines** | **communicating** | **complex** | **computer** | **data** | **datasets** | **...** | **interpreting** | **knowledge** | **large** | **learning** | **machine** | **make** | **predictions** | **science** | **statistics** | **visualization** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Document 1 | 0.000 | 0.000000 | 0.187 | 0.000000 | 0.3173 | 0.000 | 0.000000 | 0.317385 | 0.241379 | 0.000000 | ... | 0.000000 | 0.317385 | 0.000000 | 0.000000 | 0.000000 | 0.000 | 0.000000 | 0.634769 | 0.317385 | 0.000000 |
| Document 2 | 0.327 | 0.327055 | 0.193 | 0.327055 | 0.000 | 0.000 | 0.000000 | 0.000000 | 0.000000 | 0.327055 | ... | 0.000000 | 0.000000 | 0.327055 | 0.327055 | 0.327055 | 0.327 | 0.327055 | 0.000000 | 0.000000 | 0.000000 |
| Document 3 | 0.000 | 0.000000 | 0.190 | 0.000000 | 0.000 | 0.321 | 0.321704 | 0.000000 | 0.489329 | 0.000000 | ... | 0.321704 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000 | 0.000000 | 0.000000 | 0.000000 | 0.321704 |

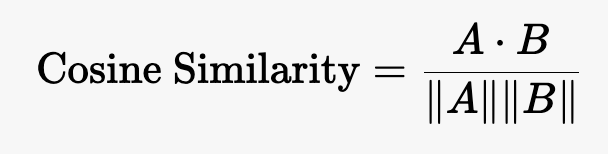
3 rows × 24 columns

**Problem 4:** Calculate Cosine Similarity

**Task**: Compute the cosine similarity between a query and the documents.

**Instructions**:

1. Define a query, for example: "data science algorithms".
2. Write a function to convert the query into a vector based on the terms in the TF-IDF matrix.
3. Implement the cosine similarity formula:



where A is the query vector and B is the document vector.

1. Rank the documents based on their cosine similarity to the query and print the results.

import pandas as pd

from sklearn**.**feature\_extraction**.**text import TfidfVectorizer

from sklearn**.**metrics**.**pairwise import cosine\_similarity

documents **=** [

**"**Data science combines statistics, computer science, and domain knowledge.**",**

**"**Machine learning algorithms can analyze large datasets and make predictions.**",**

**"**Data visualization helps in interpreting complex data and communicating insights.**"**

]

# Define the query

query **=** **"**data science algorithms**"**

# Function to compute TF-IDF matrix for documents and a query

def compute\_tfidf(documents**,** query)**:**

    # Combine documents and query into one list

    docs\_with\_query **=** documents **+** [query]

    # Initialize TfidfVectorizer and fit-transform on combined data

    vectorizer **=** TfidfVectorizer()

    tfidf\_matrix **=** vectorizer**.**fit\_transform(docs\_with\_query)

    # Separate the document and query TF-IDF vectors

    doc\_tfidf **=** tfidf\_matrix[**:-**1]  # All document vectors

    query\_tfidf **=** tfidf\_matrix[**-**1]  # Query vector

    return doc\_tfidf**,** query\_tfidf

# Function to compute cosine similarity between the query and documents

def rank\_documents\_by\_similarity(doc\_tfidf**,** query\_tfidf)**:**

    # Calculate cosine similarity between query and each document

    similarities **=** cosine\_similarity(query\_tfidf**,** doc\_tfidf)**.**flatten()

    # Rank documents by similarity

    doc\_ranking **=** sorted(enumerate(similarities**,** 1)**,** key**=**lambda x: x[1]**,** reverse**=**True)

    return doc\_ranking

# Calculate TF-IDF and cosine similarity

doc\_tfidf**,** query\_tfidf **=** compute\_tfidf(documents**,** query)

ranked\_docs **=** rank\_documents\_by\_similarity(doc\_tfidf**,** query\_tfidf)

# Display results

print(**"**Ranking of documents based on cosine similarity to the query:**"**)

for doc\_num**,** score in ranked\_docs**:**

    print(f"Document {doc\_num}: Similarity Score = {score:.4f}")

Ranking of documents based on cosine similarity to the query:

Document 1: Similarity Score = 0.4459

Document 3: Similarity Score = 0.2110

Document 2: Similarity Score = 0.1610

**Problem 5:** Advanced Query Processing and Cosine Similarity

**Task**: Preprocess documents and queries, then calculate cosine similarity with enhanced text normalization techniques.

**Instructions:**

1. Define multiple queries, for example:

* Query 1: "data scientist"
* Query 2: "machine learn"
* Query 3: "visualization of data"

1. Implement the following steps:

* **Text Normalization**: Preprocess the documents and queries by:
  + Converting text to lowercase.
  + Removing punctuation.
  + Applying stemming (using NLTK or a similar library).
* Write a Python function to:
  + Convert each preprocessed query into a vector based on the terms in the TF-IDF matrix.
  + Calculate cosine similarity for each query against all documents.
  + Rank the documents for each query based on their similarity scores and print the results.

import pandas as pd

import string

from sklearn**.**feature\_extraction**.**text import TfidfVectorizer

from sklearn**.**metrics**.**pairwise import cosine\_similarity

from nltk**.**stem**.**porter import PorterStemmer

from nltk**.**corpus import stopwords

import nltk

# Download NLTK resources

nltk**.**download(**'**stopwords**'**)

stop\_words **=** set(stopwords**.**words(**'**english**'**))

stemmer **=** PorterStemmer()

# Define the documents

documents **=** [

**"**Data science combines statistics, computer science, and domain knowledge.**",**

**"**Machine learning algorithms can analyze large datasets and make predictions.**",**

**"**Data visualization helps in interpreting complex data and communicating insights.**"**

]

# Define the queries

queries **=** [

**"**data scientist**",**

**"**machine learn**",**

**"**visualization of data**"**

]

# Function to preprocess text: lowercase, remove punctuation, apply stemming

def preprocess\_text(text)**:**

    # Lowercase and remove punctuation

    text **=** text**.**lower()**.**translate(str**.**maketrans(**'',** **'',** string**.**punctuation))

    # Tokenize and remove stopwords, then apply stemming

    tokens **=** [stemmer**.**stem(word) for word in text**.**split() if word not in stop\_words]

    return **'** **'.**join(tokens)

# Preprocess documents and queries

preprocessed\_documents **=** [preprocess\_text(doc) for doc in documents]

preprocessed\_queries **=** [preprocess\_text(query) for query in queries]

# Function to compute TF-IDF and cosine similarity

def compute\_tfidf\_and\_similarity(preprocessed\_documents**,** preprocessed\_queries)**:**

    # Combine documents and queries for TF-IDF transformation

    vectorizer **=** TfidfVectorizer()

    all\_texts **=** preprocessed\_documents **+** preprocessed\_queries

    tfidf\_matrix **=** vectorizer**.**fit\_transform(all\_texts)

    # Split TF-IDF matrix into document vectors and query vectors

    doc\_tfidf **=** tfidf\_matrix[**:**len(preprocessed\_documents)]

    query\_tfidfs **=** tfidf\_matrix[len(preprocessed\_documents)**:**]

    # Compute cosine similarity for each query against all documents

    for i**,** query\_vector in enumerate(query\_tfidfs)**:**

        similarities **=** cosine\_similarity(query\_vector**,** doc\_tfidf)**.**flatten()

        ranked\_docs **=** sorted(enumerate(similarities**,** 1)**,** key**=**lambda x: x[1]**,** reverse**=**True)

        # Print the ranking results for the query

        print(f"\nRanking for Query {i**+**1} ('{queries[i]}'):")

        for doc\_num**,** score in ranked\_docs**:**

            print(f"Document {doc\_num}: Similarity Score = {score:.4f}")

# Run the TF-IDF and similarity calculation

compute\_tfidf\_and\_similarity(preprocessed\_documents**,** preprocessed\_queries)

Ranking for Query 1 ('data scientist'):

Document 3: Similarity Score = 0.2275

Document 1: Similarity Score = 0.0990

Document 2: Similarity Score = 0.0000

Ranking for Query 2 ('machine learn'):

Document 2: Similarity Score = 0.4279

Document 1: Similarity Score = 0.0000

Document 3: Similarity Score = 0.0000

Ranking for Query 3 ('visualization of data'):

Document 3: Similarity Score = 0.5111

Document 1: Similarity Score = 0.1137

Document 2: Similarity Score = 0.0000

[nltk\_data] Downloading package stopwords to [C:\Users\Rog](file:///C:\Users\Rog)

[nltk\_data] Strix\AppData\Roaming\nltk\_data...

[nltk\_data] Package stopwords is already up-to-date!