1. Import Libraries

Code:

import json

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, precision\_score, recall\_score, f1\_score

* **json:** Used to load the processed TF-IDF data stored in JSON format.
* **numpy (np):** A library for numerical computations, used here for array manipulations.
* **train\_test\_split:** From sklearn.model\_selection, used to split the data into training and testing sets.
* **LogisticRegression:** A machine learning model used for binary classification (in this case, sentiment analysis).
* **accuracy\_score, classification\_report, confusion\_matrix, precision\_score, recall\_score, f1\_score:** Metrics to evaluate the model’s performance.

1. Load the Processed Data

Code:

json\_file\_path = 'D:/AIUB/Sentiment Analysis Research/JSON/TFIDF Hotel Reviews.json'

with open(json\_file\_path, 'r') as json\_file:

tfidf\_data = json.load(json\_file)

* **json\_file\_path:** Specifies the path to the JSON file that contains the TF-IDF data.
* **json.load(json\_file):** Reads the JSON file and converts it into a Python object (list of dictionaries) containing TF-IDF scores for each review.

1. Convert the List of Dictionaries to a Numpy Array

Code:

tfidf\_matrix = np.array([[doc.get(word, 0) for word in tfidf\_data[0].keys()] for doc in tfidf\_data])

* **np.array():** Converts the list of dictionaries (JSON format) into a 2D Numpy array.
* **doc.get(word, 0):** For each document (doc), the code retrieves the TF-IDF value for each word. If the word is not present, it assigns a default value of 0.
* **tfidf\_data[0].keys():** Uses the keys (words) from the first document in the TF-IDF data as the feature set (consistent across all documents). This creates a structured matrix where each row represents a document, and each column represents a word.

1. Prepare Labels and Split the Data

Code:

labels = np.array([1 if i < len(tfidf\_data)//2 else 0 for i in range(len(tfidf\_data))])

* **labels:** Creates a simple binary label array where the first half of the data is assigned 1 (e.g., positive sentiment) and the second half is assigned 0 (e.g., negative sentiment). This is just an example and should be replaced with actual labels if available.

1. Split the Data

Code:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(tfidf\_matrix, labels, test\_size=0.2, random\_state=42)

* **train\_test\_split():** Splits the dataset into training and testing sets.
* **tfidf\_matrix:** The feature matrix (TF-IDF values).
* **labels:** The corresponding sentiment labels (1 or 0).
* **test\_size=0.2:** Specifies that 20% of the data will be used for testing, and 80% will be used for training.
* **random\_state=42:** Ensures reproducibility of the split.

1. Check the Shape of the Splits

Code:

print(f"Training data shape: {X\_train.shape}, Training labels shape: {y\_train.shape}")

print(f"Testing data shape: {X\_test.shape}, Testing labels shape: {y\_test.shape}")

* **X\_train.shape and X\_test.shape:** Prints the dimensions of the training and testing datasets, ensuring they were split correctly.
* **y\_train.shape and y\_test.shape:** Prints the dimensions of the corresponding labels.

1. Train the Model

Code:

model = LogisticRegression()

model.fit(X\_train, y\_train)

* **LogisticRegression():** Initializes a logistic regression model.
* **model.fit(X\_train, y\_train):** Trains the model using the training data (X\_train) and corresponding labels (y\_train).

1. Predict on the Test Data

Code:

y\_pred = model.predict(X\_test)

* **model.predict(X\_test):** Uses the trained model to predict the sentiment (1 or 0) for the test dataset.

1. Evaluate the Model

Code:

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred, average='weighted')

recall = recall\_score(y\_test, y\_pred, average='weighted')

f1 = f1\_score(y\_test, y\_pred, average='weighted')

* **accuracy\_score(y\_test, y\_pred):** Calculates the accuracy of the model by comparing the true labels (y\_test) with the predicted labels (y\_pred).
* **precision\_score(y\_test, y\_pred, average='weighted'):** Calculates the precision (how many of the predicted positive instances were correct). The weighted option ensures that class imbalance is handled.
* **recall\_score(y\_test, y\_pred, average='weighted'):** Calculates the recall (how many of the true positive instances were correctly identified).
* **f1\_score(y\_test, y\_pred, average='weighted'):** Calculates the F1 score, which is the harmonic mean of precision and recall.

1. Print the Metrics

Code:

print(f"Accuracy: {accuracy \* 100:.2f}%")

print(f"Precision: {precision:.2f}")

print(f"Recall: {recall:.2f}")

print(f"F1 Score: {f1:.2f}")

* Prints the model's accuracy, precision, recall, and F1 score in a formatted way.

1. Classification Report and Confusion Matrix

Code:

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

* **classification\_report(y\_test, y\_pred):** Prints a detailed classification report showing precision, recall, F1 score, and support for each class (1 and 0).
* **confusion\_matrix(y\_test, y\_pred):** Prints the confusion matrix, showing how many instances were correctly and incorrectly classified for each class.

Summary of Workflow

1. Load the TF-IDF data from a JSON file and convert it into a structured matrix.
2. Split the data into training and testing sets.
3. Train a logistic regression model on the training set.
4. Predict the sentiment labels on the test set.
5. Evaluate the model's performance using accuracy, precision, recall, F1 score, and confusion matrix.