American International University-Bangladesh (AIUB)

Faculty of Science & Technology (FST)

Department of Computer Science

Natural Language Processing

Mid-Term Project Report

Summer 2024-2025

Section:

Group:

SL #	Student Name	Student ID	Contribution (%)
1.	Md. Tamjid Hossain	22-46460-1	35
2.	Wasif Asad Alvi	22-46451-1	35
3.	Rifat Talukdar	22-46428-1	30
4.			

Dataset Description

The dataset used in this project is the TripAdvisor Hotel Reviews dataset from Kaggle[Link: https://www.kaggle.com/datasets/andrewmvd/trip-advisor-hotel-reviews?resource=download], which contains 20,491 customer reviews collected from different hotels. Each review is accompanied by a rating on a scale of 1 to 5, where higher values generally indicate a more positive experience.

We have converted the ratings into 3 sentiment classes:

- Ratings 1,2 is converted to Negative sentiment [Class 0]
- Ratings 3,4 is converted to Neutral sentiment [Class 1]
- Rating 5 is converted to Positive sentiment [Class 2]

After performing a train-test split (70% training, 30% testing), we obtained the following class distribution:

Training set distribution:

- Positive [Class 2]: 6,337
- Neutral [Class 1]: 5,756
- Negative [Class 0]: 2,250

Testing set distribution:

- Positive [Class 2]: 2,717
- Neutral [Class 1]: 2,467
- Negative [Class 0]: 964

This shows that the dataset is imbalanced, with a larger number of positive reviews compared to negative ones. This imbalance is typical in review datasets because customers often leave more positive feedback.

Project Implementation Detail

Task 1: Importing Libraries

We imported the necessary Python libraries for data preprocessing, visualization, and model building.

Code for task1:

```
import pandas as pd
import re
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
```

Code description:

We imported required libraries for handling text preprocessing (NLTK), creating features (TF-IDF), and building a Multinomial version of Naïve Bayes classifier (sklearn).

Task 2: Downloading Necessary Components

We downloaded the required NLTK packages. This ensures all required NLTK resources are available. Without them, tokenization, stopword removal, and lemmatization would fail.

Code for task2:

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

Sample Output:

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
True
```

Figure 1: Output of Task 2

Code description:

We downloaded punkt_tab for tokenization, stopwords for removing common English stop words, wordnet for lemmatization.

Task 3: Load Dataset

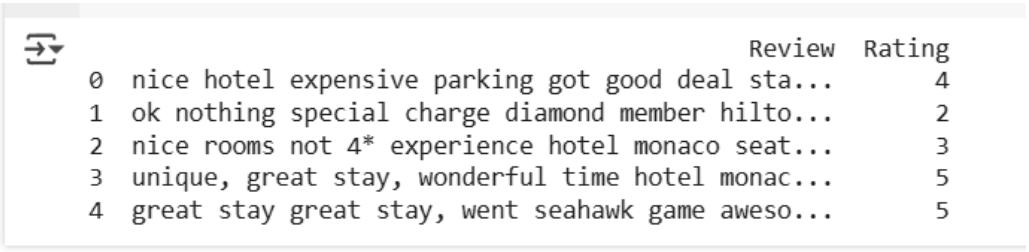
We loaded the TripAdvisor hotel reviews dataset in CSV format.

Code for task3:

```
def case_folding(text):
    return text.lower()

df['cleaned'] = df['Review'].apply(case_folding)
print(df[['Review','cleaned']].head())
```

Sample Output:



	Review	Rating
0	nice hotel expensive parking got good deal sta...	4
1	ok nothing special charge diamond member hilto...	2
2	nice rooms not 4* experience hotel monaco seat...	3
3	unique, great stay, wonderful time hotel monac...	5
4	great stay great stay, went seahawk game aweso...	5

Figure 2: Output of Task 3

Code description:

The dataset was loaded and displayed using head().

Task 4: Check DataFrame Shape and NULL Values

We checked how many rows and columns are present in the dataset and if there's any missing values or not.

Code for task4:

```
print("DataFrame Shape:", df.shape)
print("NULL Count:")
print(df.isnull().sum())
```

Sample Output:

```
DataFrame Shape: (20491, 2)
NULL Count:
Review    0
Rating    0
dtype: int64
```

Figure 3: Output of Task 4

Code description:

df.shape() shows dataset dimensions and df.isnull().sum() outputs the number of null values per column.

Task 5: Map Reviews to Sentiments

We performed mapping according to the reviews in the dataset. We mapped the rating into 3 categories-

- Positive (2): Rating = 5
- Negative (0): Ratings = 1 or 2
- Neutral (1): Ratings = 3 or 4

Code for task5:

```
pos = [5]
neg = [1,2]
neu = [3,4]

def sentiment(rating):
  if rating in pos:
    return 2
  elif rating in neg:
    return 0
  else:
    return 1
df['Sentiment'] = df['Rating'].apply(sentiment)
df.head()
```

Sample Output:

	Review	Rating	Sentiment
0	nice hotel expensive parking got good deal sta...	4	1
1	ok nothing special charge diamond member hilto...	2	0
2	nice rooms not 4* experience hotel monaco seat...	3	1
3	unique, great stay, wonderful time hotel monac...	5	2
4	great stay great stay, went seahawk game aweso...	5	2

Figure 4: Output of Task 5

Code description:

The ratings were mapped into sentiments using if else condition as demonstrated in the code.

Task 6: Case Folding

We did case folding for task 1 on our dataset. We have used the basic lower() function to remove text capitalization. Case folding converts all text to lowercase. This ensures that words like “Good” and “good” are treated the same, reducing redundancy in token representation.

Code for task6:

```
def caseFolding(text):
    return text.lower()

df['cleaned'] = df['Review'].apply(caseFolding)
print(df[['Review','cleaned']].head())
```

Sample Output:

```
                                              Review  \
0  nice hotel expensive parking got good deal sta...
1  ok nothing special charge diamond member hilto...
2  nice rooms not 4* experience hotel monaco seat...
3  unique, great stay, wonderful time hotel monac...
4  great stay great stay, went seahawk game aweso...

                                             cleaned
0  nice hotel expensive parking got good deal sta...
1  ok nothing special charge diamond member hilto...
2  nice rooms not 4* experience hotel monaco seat...
3  unique, great stay, wonderful time hotel monac...
4  great stay great stay, went seahawk game aweso...
```

Figure 5: Output of Task 6

Code description:

We applied str.lower() to each review. The lowercase version is stored in the cleaned column for further preprocessing.

Task 7: Punctuation Removal

We removed punctuation marks from our text because they do not contribute meaning in sentiment analysis. We used regular expressions to remove all non-alphanumeric characters.

Punctuation marks create noise in NLP tasks. Removing them ensures cleaner tokenization.

Code for task7:

```
def removePunctuation(text):
    return re.sub(r'[^\w\s]', '', text)

df['cleaned'] = df['cleaned'].apply(removePunctuation)
print(df[['Review','cleaned']].head())
```

Sample Output:

```
                                              Review  \
0  nice hotel expensive parking got good deal sta...
1  ok nothing special charge diamond member hilto...
2  nice rooms not 4* experience hotel monaco seat...
3  unique, great stay, wonderful time hotel monac...
4  great stay great stay, went seahawk game aweso...

                                             cleaned
0  nice hotel expensive parking got good deal sta...
1  ok nothing special charge diamond member hilto...
2  nice rooms not 4 experience hotel monaco seatt...
3  unique great stay wonderful time hotel monaco ...
4  great stay great stay went seahawk game awesom...
```

Figure 6: Output of Task 7

Code description:

We used re.sub to remove all punctuation using regex. The updated cleaned column now contains punctuation-free text.

Task 8: Tokenization

We performed tokenization to split sentences into individual words using NLTK's word_tokenize function. Tokenization splits text into individual words (tokens), which is essential for almost all NLP tasks.

Code for task8:

```
def tokenizeText(text):
    return word_tokenize(text)

df['tokens'] = df['cleaned'].apply(tokenizeText)
print(df[['cleaned','tokens']].head())
```

Sample Output:

```
                                                cleaned  \
0  nice hotel expensive parking got good deal sta...
1  ok nothing special charge diamond member hilto...
2  nice rooms not 4 experience hotel monaco seatt...
3  unique great stay wonderful time hotel monaco ...
4  great stay great stay went seahawk game awesom...

                                                tokens
0  [nice, hotel, expensive, parking, got, good, d...
1  [ok, nothing, special, charge, diamond, member...
2  [nice, rooms, not, 4, experience, hotel, monac...
3  [unique, great, stay, wonderful, time, hotel, ...
4  [great, stay, great, stay, went, seahawk, game...
```

Figure 7: Output of Task 8

Code description:

Each sentence was converted into a list of words. Tokenization allows us to process words individually for tasks like stopword removal, and lemmatization or stemming.

Task 9: Stop Words Removal

We removed common English stopwords that do not carry meaningful sentiment information using NLTK's stopwords corpus. Stop words (e.g., "is", "the", "and") usually do not help in sentiment analysis. Removing them reduces noise.

Code for task9:

```
stop_words = set(stopwords.words('english'))

def removeStopwords(tokens):
    return [word for word in tokens if word not in stop_words]

df['tokens'] = df['tokens'].apply(removeStopwords)
print(df[['cleaned','tokens']].head())
```

Sample Output:

```
                                             cleaned  \
0  nice hotel expensive parking got good deal sta...
1  ok nothing special charge diamond member hilto...
2  nice rooms not 4 experience hotel monaco seatt...
3  unique great stay wonderful time hotel monaco ...
4  great stay great stay went seahawk game awesom...

                                             tokens
0  [nice, hotel, expensive, parking, got, good, d...
1  [ok, nothing, special, charge, diamond, member...
2  [nice, rooms, 4, experience, hotel, monaco, se...
3  [unique, great, stay, wonderful, time, hotel, ...
4  [great, stay, great, stay, went, seahawk, game...
```

Figure 8: Output of Task 9

Code description:

We filtered out tokens that appear in the stopwords list. This step ensures that only meaningful words are kept for further analysis.

Task 10: Stemming

We used stemming to reduce words to their base form by removing suffixes and prefixes.

Code for task10:

```
stemmer = PorterStemmer()

def stemTokens(token):
    return [stemmer.stem(w) for w in token]
df['stemmed_tokens'] = df['tokens'].apply(stemTokens)

print(df[['tokens', 'stemmed_tokens']].head(10))
```

Sample Output:

```
                                              tokens  \
0  [nice, hotel, expensive, parking, got, good, d...
1  [ok, nothing, special, charge, diamond, member...
2  [nice, rooms, 4, experience, hotel, monaco, se...
3  [unique, great, stay, wonderful, time, hotel, ...
4  [great, stay, great, stay, went, seahawk, game...
5  [love, monaco, staff, husband, stayed, hotel, ...
6  [cozy, stay, rainy, city, husband, spent, 7, n...
7  [excellent, staff, housekeeping, quality, hote...
8  [hotel, stayed, hotel, monaco, cruise, rooms, ...
9  [excellent, stayed, hotel, monaco, past, delig...

                                      stemmed_tokens
0  [nice, hotel, expens, park, got, good, deal, s...
1  [ok, noth, special, charg, diamond, member, hi...
2  [nice, room, 4, experi, hotel, monaco, seattl,...
3  [uniqu, great, stay, wonder, time, hotel, mona...
4  [great, stay, great, stay, went, seahawk, game...
5  [love, monaco, staff, husband, stay, hotel, cr...
6  [cozi, stay, raini, citi, husband, spent, 7, n...
7  [excel, staff, housekeep, qualiti, hotel, choc...
8  [hotel, stay, hotel, monaco, cruis, room, gene...
9  [excel, stay, hotel, monaco, past, delight, re...
```

Figure 9: Output of Task 10

Code description:

We used Porter Stemmer to stem words to their root form.

Task 11: Lemmatization

We applied lemmatization to convert each word to its base form using WordNet's lemmatizer. Lemmatization reduces words to their root dictionary form (e.g., "running" → "run") using vocabulary and grammar.

Code for task11:

```
lemmatizer = WordNetLemmatizer()

def lemmatizeTokens(token):
    return [lemmatizer.lemmatize(w) for w in token]

def finalText(token):
    return ' '.join(token)

df['lemma_tokens']                                                    =
df['stemmed_tokens'].apply(lemmatizeTokens)
df['final_text'] = df['lemma_tokens'].apply(finalText)
df['final_text'] = df['final_text'].apply(caseFolding)

print(df[['stemmed_tokens', 'lemma_tokens']].head(10))
```

Sample Output:

```
                                      stemmed_tokens  \
0  [nice, hotel, expens, park, got, good, deal, s...
1  [ok, noth, special, charg, diamond, member, hi...
2  [nice, room, 4, experi, hotel, monaco, seattl,...
3  [uniqu, great, stay, wonder, time, hotel, mona...
4  [great, stay, great, stay, went, seahawk, game...
5  [love, monaco, staff, husband, stay, hotel, cr...
6  [cozi, stay, raini, citi, husband, spent, 7, n...
7  [excel, staff, housekeep, qualiti, hotel, choc...
8  [hotel, stay, hotel, monaco, cruis, room, gene...
9  [excel, stay, hotel, monaco, past, delight, re...

                                       lemma_tokens
0  [nice, hotel, expens, park, got, good, deal, s...
1  [ok, noth, special, charg, diamond, member, hi...
2  [nice, room, 4, experi, hotel, monaco, seattl,...
3  [uniqu, great, stay, wonder, time, hotel, mona...
4  [great, stay, great, stay, went, seahawk, game...
5  [love, monaco, staff, husband, stay, hotel, cr...
6  [cozi, stay, raini, citi, husband, spent, 7, n...
7  [excel, staff, housekeep, qualiti, hotel, choc...
8  [hotel, stay, hotel, monaco, cruis, room, gene...
9  [excel, stay, hotel, monaco, past, delight, re...
```

Figure 10: Output of Task 11

Code description:

We lemmatized each token to its base form. Then we converted token lists back to sentences for vectorization.

Task 12: Vector Semantics (TF-IDF)

We converted our cleaned text into numerical features using TF-IDF. This allows our model to process text. TF-IDF assigns weights to words based on their importance in a document relative to the whole corpus.

Code for task12:

```
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(df['final_text'])
y = df['Sentiment']

corpus = df['final_text'].tolist()
tfidf_vector = TfidfVectorizer(
    smooth_idf=False,
    token_pattern=r'\b[a-zA-Z]{2,}\b',
    max_features=20
)

tfidf_matrix = tfidf_vector.fit_transform(corpus)
df_tfidf = pd.DataFrame(tfidf_matrix.toarray(),
columns=tfidf_vector.get_feature_names_out())
df_tfidf.head()
```

Sample Output:

	beach	breakfast	clean	day	good	great	hotel	like	locat	nice	night
0	0.0	0.000000	0.147442	0.000000	0.138058	0.125083	0.186560	0.172658	0.136314	0.764620	0.285329
1	0.0	0.370265	0.116419	0.125476	0.436041	0.098765	0.515574	0.272661	0.000000	0.120748	0.225295
2	0.0	0.000000	0.000000	0.000000	0.085765	0.233115	0.173844	0.000000	0.000000	0.285001	0.443135
3	0.0	0.000000	0.000000	0.000000	0.000000	0.444018	0.441496	0.408599	0.161295	0.361897	0.000000
4	0.0	0.000000	0.000000	0.000000	0.112995	0.409504	0.229038	0.000000	0.111568	0.000000	0.000000

Figure 11: Output of Task 12

Code description:

TfidfVectorizer() converts text into TF-IDF features. Here, smooth_idf=False disables smoothing of inverse document frequency; the raw IDF is used. We kept max_features=20 to keep only the top 20 most frequent words in the vocabulary to show the TF-IDF matrix.

Task 13: Plot histogram of TF-IDF sums

We plotted a histogram using the sum of TF-IDF values per document

Code for task13:

```
tfidf_sums = df_tfidf.sum(axis=1)
print("Vocabulary size:", len(vectorizer.vocabulary_))

plt.figure(figsize=(8,5))
plt.hist(tfidf_sums)
plt.xlabel("Sum of TF-IDF values per document")
plt.ylabel("Number of Documents")
plt.title("Distribution of TF-IDF Sum per Document")
plt.show()
```

Sample Output:

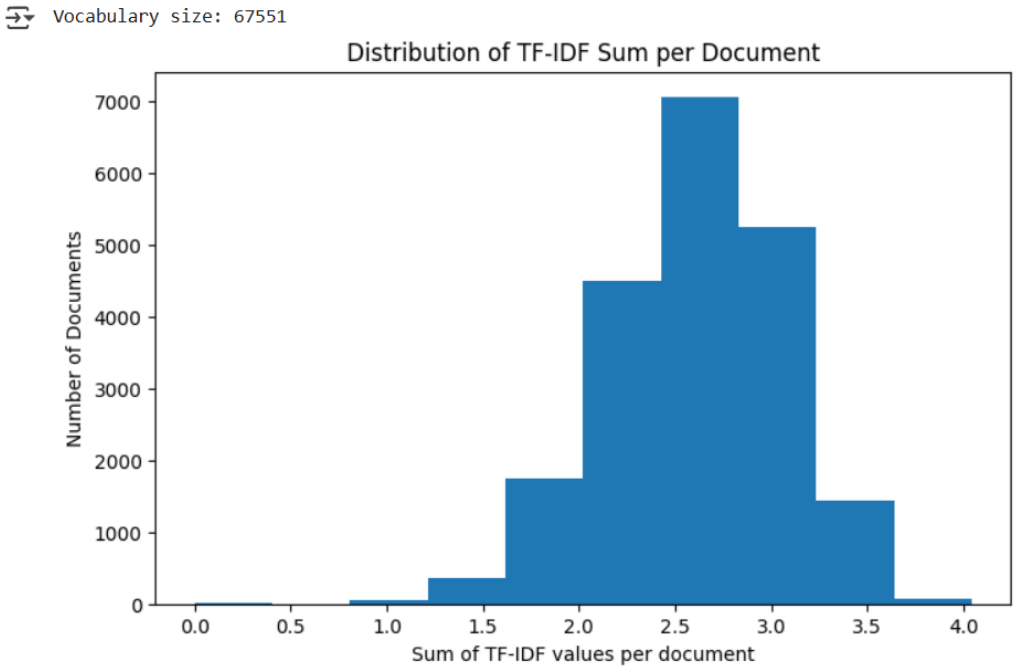


Figure 12: Output of Task 13

Code description:

len(vectorizer.vocabulary_) returns the total vocbulary size of TF-IDF matrix. df_tfidf.sum(axis=1) returns the TF-IDF sum per document. We used matplotlib to visualize the histogram.

Task 14: Train-Test Split

We split the dataset into training (70%) and testing (30%) sets to train a Naive Bayes classifier.

Code for task14:

```
X_train , X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42, stratify=y)
print(y_train.value_counts())
print(y_test.value_counts())
```

Sample Output:

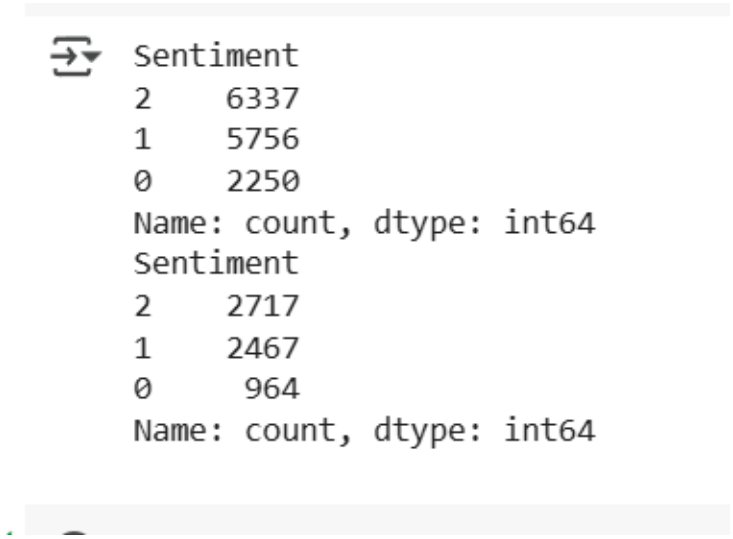


Figure 13: Output of Task 14

Code description:

We used train_test_split with stratification to maintain class distribution. Trained MultinomialNB on TF-IDF features. Predicted sentiments on the test set.

Task 15: Model training

We used the split training dataset to train a Multinomial Naïve Bayes model.

Code for task15:

```
model = MultinomialNB()
model.fit(X_train, y_train)
```

Sample Output:

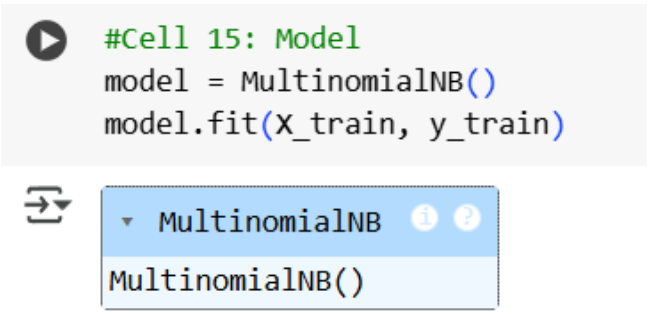


Figure 14: Output of Task 15

Code description:

We trained MultinomialNB on TF-IDF features. Predicted sentiments on the test set.

Task 16: Model Evaluation

We evaluated the model using accuracy, precision, recall and F1-score.

Code for task16:

```
y_pred = model.predict(X_test)

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')

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}")
```

Sample Output:

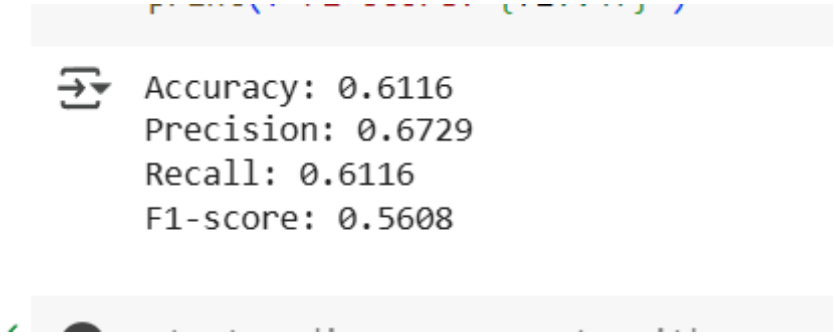


Figure 15: Output of Task 16

Code description:

We used model.predict(X_test) to make predictions on the test reviews. Then the prediction is stored into y_pred. We used the prediction to find the accuracy, precision, recall and f1_score. These scores represent how accurate the model is to predict sentiments.

Project Code

```
import pandas as pd
import re
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score

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

df = pd.read_csv("/tripadvisor_hotel_reviews.csv")
print(df.head())

print("DataFrame Shape:", df.shape)
print("NULL Count:")
print(df.isnull().sum())

pos = [5]
neg = [1,2]
neu = [3,4]

def sentiment(rating):
  if rating in pos:
    return 2
  elif rating in neg:
    return 0
  else:
    return 1
df['Sentiment'] = df['Rating'].apply(sentiment)
df.head()

def caseFolding(text):
    return text.lower()

df['cleaned'] = df['Review'].apply(caseFolding)
print(df[['Review','cleaned']].head())

def removePunctuation(text):
    return re.sub(r'[^\w\s]', '', text)

df['cleaned'] = df['cleaned'].apply(removePunctuation)
print(df[['Review','cleaned']].head())

def tokenizeText(text):
    return word_tokenize(text)

df['tokens'] = df['cleaned'].apply(tokenizeText)
print(df[['cleaned','tokens']].head())
```

```
stop_words = set(stopwords.words('english'))

def removeStopwords(tokens):
    return [word for word in tokens if word not in stop_words]

df['tokens'] = df['tokens'].apply(removeStopwords)
print(df[['cleaned','tokens']].head())

stemmer = PorterStemmer()

def stemTokens(token):
    return [stemmer.stem(w) for w in token]
df['stemmed_tokens'] = df['tokens'].apply(stemTokens)

print(df[['tokens', 'stemmed_tokens']].head(10))

lemmatizer = WordNetLemmatizer()

def lemmatizeTokens(token):
    return [lemmatizer.lemmatize(w) for w in token]

def finalText(token):
    return ' '.join(token)

df['lemma_tokens']                                                    =
df['stemmed_tokens'].apply(lemmatizeTokens)

df['final_text'] = df['lemma_tokens'].apply(finalText)
df['final_text'] = df['final_text'].apply(caseFolding)

print(df[['stemmed_tokens', 'lemma_tokens']].head(10))

vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(df['final_text'])
y = df['Sentiment']

corpus = df['final_text'].tolist()
tfidf_vector = TfidfVectorizer(
    smooth_idf=False,
    token_pattern=r'\b[a-zA-Z]{2,}\b',
    max_features=20
)

tfidf_matrix = tfidf_vector.fit_transform(corpus)
df_tfidf         =         pd.DataFrame(tfidf_matrix.toarray(),
columns=tfidf_vector.get_feature_names_out())
df_tfidf.head()

tfidf_sums = df_tfidf.sum(axis=1)
print("Vocabulary size:", len(vectorizer.vocabulary_))

plt.figure(figsize=(8,5))
plt.hist(tfidf_sums)
plt.xlabel("Sum of TF-IDF values per document")
plt.ylabel("Number of Documents")
plt.title("Distribution of TF-IDF Sum per Document")
plt.show()

X_train,  X_test,  y_train,  y_test  =  train_test_split(X,  y,
test_size=0.3, random_state=42, stratify=y)
print(y_train.value_counts())
```

```
print(y_test.value_counts())

model = MultinomialNB()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

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')

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}")
```