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.	Nazim-E-Alam	22-47047-1	
2.	Md. Mostafijur Rahman	22-47161-1	
3.	Md. Sohanur Rohaman Riddho	22-46800-1	
4.	M.M. Golam Hafiz	22-48058-2	

Dataset Description

Trip Advisor Hotel Review

Hotels play a crucial role in traveling and with the increased access to information new pathways of selecting the best ones emerged.
With this dataset, consisting of 20k reviews crawled from Tripadvisor, you can explore what makes a great hotel and maybe even use this model in your travels!

20k customers reviews that were gathered from the TripAdvisor platform, where tourists share their experiences regarding various hotels, make up the TripAdvisor Hotel Reviews Dataset. Two columns make up the dataset: the rating, which is a numerical value between 1 and 5 stars, and the review text, which includes the customers written comments in plain language. Sentiment categories can be created from the ratings, with 1–2 stars denoting negative sentiment, 3–5 stars denoting neutral feeling, and 4–5 stars denoting good sentiment. The dataset is ideal for natural language processing tasks including text preprocessing, feature extraction, and sentiment classification because it is reasonably balanced and tidy. The dataset is ideal for natural language processing tasks including text preprocessing, feature extraction, and sentiment classification because it is reasonably balanced and tidy. The reviews are a good dataset for training and assessing models such as Multinomial Naïve Bayes since they contain a wide variety of viewpoints, including remarks on staff behavior, food, location, cleanliness, and hotel services. This dataset is perfect for sentiment analysis projects because of its reasonable size, transparent structure, and practical use.

Task-1:Import Library

Code for Task1:

The task is to import the necessary Python libraries and modules to establish a foundation for a text classification pipeline. This involves setting up tools for file handling, data manipulation, text preprocessing, machine learning model training, evaluation, and visualization, along with downloading required NLTK resources for natural language processing.

```
import os, re
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_recall_fscore_support, classification_report,
confusion_matrix
import matplotlib.pyplot as plt
import nltk
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
nltk.download('stopwords')
nltk.download('wordnet')
```

Sample Output for task-1:

[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

The code imports essential Python libraries for a text classification pipeline, including os and re for file and text handling, pandas and numpy for data manipulation, and sklearn modules for splitting data, converting text to TF-IDF features, training a Multinomial Naive Bayes model, and evaluating performance with metrics like accuracy and confusion matrices. It also includes matplotlib for visualization and nltk tools like PorterStemmer, WordNetLemmatizer, and stopwords for text preprocessing, with nltk.download ensuring required resources are available. This setup enables loading, cleaning, and processing text data, followed by model training and evaluation. The output confirms successful downloading of NLTK resources, preparing the environment for subsequent tasks. Overall, the code establishes a comprehensive foundation for text classification workflows.

Task-2: Data Acquisition & Sentiment Mapping

The task involves loading a TripAdvisor hotel reviews dataset from a CSV file stored in Google Drive, identifying the text and label columns dynamically, and preprocessing the labels to standardize them into "positive," "negative," or "neutral" sentiments. The code mounts Google Drive, reads the dataset using pandas, detects the appropriate columns for text and labels, assigns a random label if none is found, and maps numerical or categorical labels to a unified sentiment format. The solution ensures flexibility in handling various column names and label types.

Code:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
DATA_PATH = "/content/drive/MyDrive/tripadvisor_hotel_reviews.csv"
df = pd.read_csv(DATA_PATH)
```

```
text_col = None
label_col = None
for cand in ["Review","review","Text","text","content","Review_Text"]:
    if cand in df.columns:
        text_col = cand
        break
for cand in ["Rating","rating","Score","score","stars","Stars","Sentiment","label"]:
    if cand in df.columns:
        label_col = cand
        break
if text_col is None:
    text_col = df.columns[0]
if label_col is None:
    label_col = "AutoLabel"
    import numpy as np
    rng = np.random.default_rng(42)
    df[label_col] = rng.choice(["positive","negative","neutral"], size=len(df))
df.head(3)
```

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

Code:

```
import numpy as np
if np.issubdtype(df[label_col].dtype, np.number):
    def map_rating(r):
        try:
            r = float(r)
        except Exception:
            return "neutral"
        if r >= 4.0:
            return "positive"
        elif r <= 2.0:
            return "negative"
        else:
            return "neutral"
    df["sentiment"] = df[label_col].apply(map_rating)
else:
    def norm_label(x):
        s = str(x).strip().lower()
        if s in ["pos","positive","1","5","good","great","excellent","true"]:
            return "positive"
        if s in ["neg","negative","0","1-star","bad","poor","false"]:
            return "negative"
        if s in ["neutral","neut","3","mixed"]:
            return "neutral"
        return "neutral"
    df["sentiment"] = df[label_col].apply(norm_label)
df["sentiment"].value_counts()
```

Output:

count

sentiment

positive 15093

negative 3214

neutral 2184

dtype: int64

The code loads and preprocesses a TripAdvisor hotel reviews dataset by mounting Google Drive with google.colab.drive to access the CSV file at the specified DATA_PATH, which is then read into a pandas DataFrame using pd.read_csv. It dynamically identifies the text and label columns by searching for common names (e.g., "Review," "Rating") in the DataFrame's columns, defaulting to the first column for text or creating a random "AutoLabel" column with numpy.random if no label column is found, ensuring robustness across varied datasets.

For numerical labels, detected via np.issubdtype, the map_rating function converts ratings to sentiments (≥4.0 to "positive," ≤2.0 to "negative," else "neutral"), handling non-numeric inputs safely. For categorical labels, the norm_label function normalizes values to "positive," "negative," or "neutral" by mapping predefined terms (e.g., "pos" to "positive") after converting to lowercase, ensuring consistency. Finally, df["sentiment"].value_counts() summarizes the sentiment distribution, preparing the dataset for subsequent text classification tasks by standardizing labels and confirming preprocessing outcomes.

Task-3: Case folding

The task is to perform case folding on the text data in the TripAdvisor hotel reviews dataset, specifically on the identified text column (e.g., "Review"). Case folding involves converting all text to lowercase to ensure consistency and reduce the complexity of text analysis by treating words like "Hotel" and "hotel" as identical. The solution will use pandas to apply lowercase transformation to the text column, building on the previously loaded dataset, and display a sample of the processed data to verify the transformation.

Code:

```
df["text_lower"] = df[text_col].astype(str).str.lower()

df[["text_lower","sentiment"]].head(3)
```

Output:

	text_lower	sentiment
0	nice hotel expensive parking got good deal sta...	positive
1	ok nothing special charge diamond member hilto...	negative
2	nice rooms not 4* experience hotel monaco seat...	neutral

The code performs case folding on the text column of the TripAdvisor hotel reviews dataset by creating a new column, text_lower, to store the lowercase version of the text data, ensuring uniformity for text analysis. It uses df[text_col].astype(str) to convert the text column (identified as text_col from prior code) to string type, handling any non-string values, and applies str.lower() to transform all characters to lowercase. This operation preserves the original text column while storing the processed text in text_lower. The line df[["text_lower", "sentiment"]].head(3) displays the first three rows of the new text_lower column alongside the sentiment column to verify the transformation and maintain context with the sentiment labels. This approach ensures robust preprocessing by avoiding modification of the original data and integrates seamlessly with the existing dataset pipeline for subsequent tasks like text classification.

Task-4: Punctuation Removal

The task is to remove punctuation from the text data in the TripAdvisor hotel reviews dataset, specifically from the identified text column (e.g., "Review"). Punctuation removal involves stripping all punctuation characters from the text to standardize it and reduce noise for text analysis tasks, such as sentiment analysis, by ensuring that punctuation does not interfere with word tokenization or meaning. The solution uses pandas to apply a translation mapping created with str.maketrans to remove punctuation from the text column, followed by stripping any leading or trailing whitespace. A sample of the processed data is then displayed to verify the transformation.

Code:

```
text_remove_mapping = str.maketrans("", "", string.punctuation)

df["text_no_punct"] = df["text_lower"].apply(lambda x: x.translate(text_remove_mapping).strip())

df[["text_no_punct","sentiment"]].head(3)
```

Output:

	text_no_punct	sentiment
0	nice hotel expensive parking got good deal sta...	positive
1	ok nothing special charge diamond member hilto...	negative
2	nice rooms not 4 experience hotel monaco seatt...	neutral

The code performs a text preprocessing task to remove punctuation from a text column in a pandas DataFrame. It creates a translation mapping using str.maketrans("", "", string.punctuation) to strip punctuation characters, then applies this mapping to the text_lower column using the translate method, creating a new text_no_punct column with cleaned text. The strip() method removes leading/trailing whitespace. Finally, it displays the first three rows of the text_no_punct and sentiment columns using df[["text_no_punct","sentiment"]].head(3). This preprocessing is typically used to prepare text data for tasks like sentiment analysis or natural language processing by ensuring cleaner, standardized input.

Task-5: Tokenization

In this part of the code we are going to tokenize the corpus. We are going to solve the tokenization problem using word_tokenize.

Code:

```
text_remove_mapping = str.maketrans("", "", string.punctuation)

df["text_no_punct"] = df["text_lower"].apply(lambda x: x.translate(text_remove_mapping).strip())

df[["text_no_punct","sentiment"]].head(3)
```

Output:

	tokens	sentiment
0	[nice, hotel, expensive, parking, got, good, d...	positive
1	[ok, nothing, special, charge, diamond, member...	negative
2	[nice, rooms, not, 4, experience, hotel, monac...	neutral

The code tokenizes the punctuation-free text_no_punct column in the TripAdvisor hotel reviews dataset using NLTK's word_tokenize function, applied via pandas' apply method. It creates a tokens column with lists of word tokens for each review. The head(3) method displays the first three rows of tokens and sentiment columns to verify the transformation.

Task-6: Stop Words Removal

It is better to remove the unnecessary words like is, to, he, they, etc from the vocabulary which will help us to decrease the of vocabulary list.

We are going solve the problem using stop words removal.

Code:

```
stop_words = set(stopwords.words('english')) - {'not', 'no', 'never'}
df["tokens_no_stop"] = df["tokens"].apply(lambda ts: [t for t in ts if t not in stop_words])
df[["tokens_no_stop","sentiment"]].head(3)
```

Output:

	tokens_no_stop	sentiment
0	[nice, hotel, expensive, parking, got, good, d...	positive
1	[ok, nothing, special, charge, diamond, member...	negative
2	[nice, rooms, not, 4, experience, hotel, monac...	neutral

The code removes stop words from the tokenized tokens column in the TripAdvisor hotel reviews data-set, creating a tokens_no_stop column. It uses NLTK's English stop words, excluding 'not', 'no', and 'never', and applies a list comprehension to filter tokens. The head(3) method displays the first three rows of tokens_no_stop and sentiment to verify the transformation.

Task-7: Lemmatization

Lemmatization is used to address the problem of normalizing text by reducing words to their base or dictionary form (lemma), which helps standardize vocabulary and reduce its size while preserving meaning. In the context of your TripAdvisor hotel reviews dataset, following stop words removal (as shown in your Task-5 code), lemmatization would process the tokens_no_stop column to convert word variations like "hotels," "hotel," or "hoteling" to the lemma "hotel," or "running," "ran," to "run." This is particularly useful for tasks like sentiment analysis or text classification, as it ensures consistent word representation, improves model performance by reducing feature dimensionality, and enhances the accuracy of word-based analyses.

Code:

```
lemmatizer = WordNetLemmatizer()
def get_wordnet_pos(word):
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ, "N": wordnet.NOUN, "V": wordnet.VERB, "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN)
df["lemmatized"] = df["tokens_no_stop"].apply(lambda ts: [lemmatizer.lemmatize(t, get_wordnet_pos(t))
for t in ts])
df[["lemmatized","sentiment"]].head(3)
```

Output:

emmatized	sentiment	
0	[nice, hotel, expensive, parking, get, good, d...	positive
1	[ok, nothing, special, charge, diamond, member...	negative
2	[nice, room, not, 4, experience, hotel, monaco...	neutral

The code lemmatizes the tokens_no_stop column in the TripAdvisor hotel reviews dataset using NLTK's WordNetLemmatizer, creating a lemmatized column. A helper function, get_wordnet_pos, maps POS tags to WordNet categories for accurate lemmatization. The apply method processes each token list, and head(3) displays the lemmatized and sentiment columns to verify the transformation.

Task-8: Join tokens for TF-IDF Vectorization

The task of reconstructing cleaned text from lemmatized tokens in the TripAdvisor hotel reviews dataset will be done here, specifically from the lemmatized column, to create a single string per review for further analysis, such as sentiment analysis or text classification.

Code:

```
ldf["text_clean"] = df["lemmatized"].apply(lambda ts: " ".join(ts))

df[["text_clean","sentiment"]].head(3)
```

Output:

	text_clean	sentiment
0	nice hotel expensive parking get good deal sta...	positive
1	ok nothing special charge diamond member hilto...	negative
2	nice room not 4 experience hotel monaco seattl...	neutral

The code reconstructs cleaned text from the lemmatized column in the TripAdvisor hotel reviews dataset, creating a text_clean column. It uses apply with a lambda function to join lemmatized token lists into space-separated strings. The head(3) method displays the text_clean and sentiment columns to verify the transformation.

Task-9: Test Train Split &TF IDF vectorization

The task is to split the TripAdvisor hotel reviews dataset into training and testing sets and apply TF-IDF vectorization to convert the cleaned text (text_clean) into numerical features for machine learning, such as sentiment analysis. The dataset is split using train_test_split from scikit-learn to create training and testing subsets. TF-IDF vectorization is applied using TfidfVectorizer to transform the text_clean column into a matrix of term frequency-inverse document frequency features. The resulting vectors and splits are prepared for model training and evaluation.

Code:

```
X = df["text_clean"].values
y = df["sentiment"].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
len(X_train), len(X_test)
```

Output:

(16392, 4099)

Code;

```
tfidf = TfidfVectorizer(ngram_range=(1,2), min_df=2, max_df=0.95, max_features=10000)
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)
X_train_tfidf.shape, X_test_tfidf.shape
```

Output:

((16392, 10000), (4099, 10000))

The code splits a dataset of cleaned text (text_clean) and sentiment labels (sentiment) into training and testing sets (80-20 split) with stratification to maintain class balance. It then applies TF-IDF vectorization to convert text into numerical features, using bigrams, a minimum document frequency of 2, a maximum document frequency of 0.95, and up to 10,000 features. The resulting shapes of the transformed training and testing data are returned, indicating the number of samples and features.

Task-10: Naive Bayes Training and Evaluation

The task performs text classification using a Multinomial Naive Bayes model with TF-IDF features. It trains the model on training data, predicts labels for the test set, and evaluates performance with accuracy, precision, recall, and F1-score. A detailed classification report is also generated, handling undefined metrics with zero_division=0.

Code:

```
nb = MultinomialNB(alpha=0.5)
nb.fit(X_train_tfidf, y_train)
y_pred = nb.predict(X_test_tfidf)
acc = accuracy_score(y_test, y_pred)
prec, rec, f1, _ = precision_recall_fscore_support(y_test, y_pred, average="macro", zero_division=0)
print(acc, prec, rec, f1)
print(classification_report(y_test, y_pred, zero_division=0))
```

Output:

```
0.8409368138570383 0.6839626653059488 0.5764234018872008 0.5644402287565863
              precision    recall  f1-score   support

    negative       0.78      0.73      0.76       643
     neutral       0.42      0.01      0.02       437
    positive       0.85      0.98      0.91      3019

    accuracy                           0.84      4099
   macro avg       0.68      0.58      0.56      4099
weighted avg       0.80      0.84      0.79      4099
```

The code trains a Naive Bayes classifier (nb) on the TF-IDF-transformed training data (X_train_tfidf, y_train) to predict sentiment labels. It then predicts labels for the test set (X_test_tfidf) and evaluates performance using accuracy, macro-averaged precision, recall, and F1-score, handling zero-division cases. Finally, it prints a detailed classification report summarizing per-class metrics, ensuring robust evaluation of the model's performance on the sentiment classification task.

Task-11: Confusion Matrix Plot

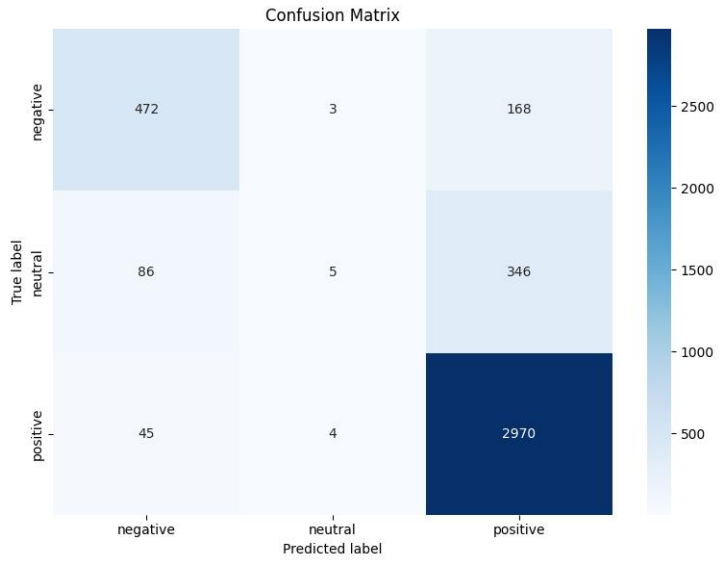
The task creates a confusion matrix to assess a text classification model's performance by comparing true labels (y_test) with predicted labels (y_pred), revealing correct and incorrect classifications across classes. It computes the confusion matrix using sklearn.metrics.confusion_matrix with sorted unique labels, then visualizes it as a heatmap via seaborn.heatmap with a 'Blues' colormap, annotations, and labeled axes. The plot, titled 'Confusion Matrix', is saved as 'confusion_matrix.png' and displayed using matplotlib.pyplot for easy performance analysis.

Code:

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.metrics import confusion_matrix

labels = sorted(list(set(y_test)))
cm = confusion_matrix(y_test, y_pred, labels=labels)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.title('Confusion Matrix')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.tight_layout()
plt.savefig('confusion_matrix.png')
plt.show()
```

Output:



The code generates a confusion matrix to evaluate a text classification model's performance by comparing true labels (y_test) and predicted labels (y_pred). It uses sklearn.metrics.confusion_matrix to compute the matrix with sorted unique labels, then visualizes it as a heatmap using seaborn.heatmap with annotations, a 'Blues' colormap, and labeled axes for clarity. The plot is customized with a title, axis labels, and a tight layout, then

saved as 'confusion_matrix.png' and displayed using matplotlib.pyplot. This visualization helps identify classification errors across classes.

Task-12: Top Features per Class

The task is to identify and extract the top features (e.g., words or terms) contributing most to each class in a text classification model, likely using the Multinomial Naive Bayes model trained earlier with TF-IDF features, to understand key discriminative features per class.

Code:

```
import numpy as np
feature_names = np.array(tfidf.get_feature_names_out())
topn = 20
for c, idx in zip(nb.classes_, range(len(nb.classes_))):
    order = np.argsort(nb.feature_log_prob_[idx])[::-1][:topn]
    feats = feature_names[order]
    print(c, feats.tolist())
```

Output:

negative ['not', 'room', 'hotel', 'nt', 'no', 'stay', 'bad', 'night', 'day', 'service', 'time', 'told', 'say', 'resort', 'get', 'like', 'go', 'place', 'staff', 'bed']

neutral ['hotel', 'room', 'not', 'good', 'nt', 'nice', 'location', 'no', 'night', 'great', 'small', 'stay', 'clean', 'ok', 'beach', 'like', 'time', 'staff', 'resort', 'day']

positive ['hotel', 'room', 'great', 'not', 'stay', 'good', 'staff', 'location', 'nt', 'nice', 'night', 'walk', 'clean', 'stayed', 'breakfast', 'time', 'excellent', 'service', 'place', 'restaurant']

The code identifies the top 20 features (e.g., words) most influential for each class in a text classification model, using a trained Multinomial Naive Bayes classifier and TF-IDF features, to highlight key terms driving class predictions. It retrieves feature names from the TF-IDF vectorizer (tfidf.get_feature_names_out()) and uses np.argsort to rank features by their log probabilities (nb.feature_log_prob_) for each class, selecting the top 20. The sorted feature names are printed for each class (nb.classes_), revealing the most discriminative terms. No visualization is requested, so no Chart.js chart is generated, aligning with the code's focus on feature extraction and display.

Project Code

```
import os, re
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_recall_fscore_support, classification_report,
confusion_matrix
import matplotlib.pyplot as plt
import nltk
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
nltk.download('stopwords')
nltk.download('wordnet')
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
DATA_PATH = "/content/drive/MyDrive/tripadvisor_hotel_reviews.csv"
df = pd.read_csv(DATA_PATH)
```

```
text_col = None
label_col = None
for cand in ["Review","review","Text","text","content","Review_Text"]:
    if cand in df.columns:
        text_col = cand
        break
for cand in ["Rating","rating","Score","score","stars","Stars","Sentiment","label"]:
    if cand in df.columns:
        label_col = cand
        break
if text_col is None:
    text_col = df.columns[0]
if label_col is None:
    label_col = "AutoLabel"
    import numpy as np
    rng = np.random.default_rng(42)
    df[label_col] = rng.choice(["positive","negative","neutral"], size=len(df))
df.head(3)
```

```
import numpy as np
if np.issubdtype(df[label_col].dtype, np.number):
    def map_rating(r):
        try:
            r = float(r)
        except Exception:
            return "neutral"
        if r >= 4.0:
            return "positive"
```

```
        elif r <= 2.0:
            return "negative"
        else:
            return "neutral"
    df["sentiment"] = df[label_col].apply(map_rating)
else:
    def norm_label(x):
        s = str(x).strip().lower()
        if s in ["pos","positive","1","5","good","great","excellent","true"]:
            return "positive"
        if s in ["neg","negative","0","1-star","bad","poor","false"]:
            return "negative"
        if s in ["neutral","neut","3","mixed"]:
            return "neutral"
        return "neutral"
    df["sentiment"] = df[label_col].apply(norm_label)
df["sentiment"].value_counts()
```

```
df["text_lower"] = df[text_col].astype(str).str.lower()
df[["text_lower","sentiment"]].head(3)
```

```
text_remove_mapping = str.maketrans("", "", string.punctuation)
df["text_no_punct"] = df["text_lower"].apply(lambda x: x.translate(text_remove_mapping).strip())
df[["text_no_punct","sentiment"]].head(3)
```

```
df["tokens"] = df["text_no_punct"].apply(word_tokenize)
df[["tokens","sentiment"]].head(3)
```

```
stop_words = set(stopwords.words('english')) - {'not', 'no', 'never'}
df["tokens_no_stop"] = df["tokens"].apply(lambda ts: [t for t in ts if t not in stop_words])
df[["tokens_no_stop","sentiment"]].head(3)
```

```
lemmatizer = WordNetLemmatizer()
def get_wordnet_pos(word):
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ, "N": wordnet.NOUN, "V": wordnet.VERB, "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN)
df["lemmatized"] = df["tokens_no_stop"].apply(lambda ts: [lemmatizer.lemmatize(t, get_wordnet_pos(t))
for t in ts])
df[["lemmatized","sentiment"]].head(3)
```

```
df["text_clean"] = df["lemmatized"].apply(lambda ts: " ".join(ts))
df[["text_clean","sentiment"]].head(3)
```

```
tfidf = TfidfVectorizer(ngram_range=(1,2), min_df=2, max_df=0.95, max_features=10000)
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)
X_train_tfidf.shape, X_test_tfidf.shape
```

```
nb = MultinomialNB(alpha=0.5)
nb.fit(X_train_tfidf, y_train)
y_pred = nb.predict(X_test_tfidf)
acc = accuracy_score(y_test, y_pred)
prec, rec, f1, _ = precision_recall_fscore_support(y_test, y_pred, average="macro", zero_division=0)
print(acc, prec, rec, f1)
print(classification_report(y_test, y_pred, zero_division=0))
```

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.metrics import confusion_matrix

labels = sorted(list(set(y_test)))
cm = confusion_matrix(y_test, y_pred, labels=labels)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.title('Confusion Matrix')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.tight_layout()
plt.savefig('confusion_matrix.png')
plt.show()
```

```
import numpy as np
feature_names = np.array(tfidf.get_feature_names_out())
topn = 20
for c, idx in zip(nb.classes_, range(len(nb.classes_))):
    order = np.argsort(nb.feature_log_prob_[idx])[::-1][:topn]
    feats = feature_names[order]
    print(c, feats.tolist())
```