# HW 1 Sentiment Analysis - HW Report as Jupyter Notebook pdf Format

```
%pip install nltk
%pip install beautifulsoup4
%pip install contractions
%pip install bs4
%pip install scikit-learn

Requirement already satisfied: nltk in c:\users\krusa\appdata\local\
packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\
localcache\local-packages\python311\site-packages (3.8.1)
Requirement already satisfied: click in c:\users\krusa\appdata\local\
packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\
localcache\local-packages\python311\site-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in c:\users\krusa\appdata\local\
packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\
localcache\local-packages\python311\site-packages (from nltk) (1.3.2)
Requirement already satisfied: regex>=2021.8.3 in c:\users\krusa\
appdata\local\packages\
pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from nltk) (2023.8.8)
Requirement already satisfied: tqdm in c:\users\krusa\appdata\local\
packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\
localcache\local-packages\python311\site-packages (from nltk) (4.66.1)
Requirement already satisfied: colorama in c:\users\krusa\appdata\
local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\
localcache\local-packages\python311\site-packages (from click->nltk)
(0.4.6)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: beautifulsoup4 in c:\users\krusa\
appdata\local\packages\
pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (4.12.2)Note: you may need to restart
the kernel to use updated packages.

Requirement already satisfied: soupsieve>1.2 in c:\users\krusa\
appdata\local\packages\
pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from beautifulsoup4) (2.5)
Requirement already satisfied: contractions in c:\users\krusa\appdata\
local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\
localcache\local-packages\python311\site-packages (0.1.73)
Requirement already satisfied: textsearch>=0.0.21 in c:\users\krusa\
appdata\local\packages\
pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from contractions) (0.0.24)
Requirement already satisfied: anyascii in c:\users\krusa\appdata\
```

```
local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\
localcache\local-packages\python311\site-packages (from
textsearch>=0.0.21->contractions) (0.3.2)
Requirement already satisfied: pyahocorasick in c:\users\krusa\
appdata\local\packages\
pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from textsearch>=0.0.21-
>contractions) (2.0.0)
Note: you may need to restart the kernel to use updated packages.
Collecting bs4
  Downloading bs4-0.0.1.tar.gz (1.1 kB)
  Installing build dependencies: started
  Installing build dependencies: finished with status 'done'
  Getting requirements to build wheel: started
  Getting requirements to build wheel: finished with status 'done'
  Preparing metadata (pyproject.toml): started
  Preparing metadata (pyproject.toml): finished with status 'done'
Requirement already satisfied: beautifulsoup4 in c:\users\krusa\
appdata\local\packages\
pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from bs4) (4.12.2)
Requirement already satisfied: soupsieve>1.2 in c:\users\krusa\
appdata\local\packages\
pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from beautifulsoup4->bs4) (2.5)
Building wheels for collected packages: bs4
  Building wheel for bs4 (pyproject.toml): started
  Building wheel for bs4 (pyproject.toml): finished with status 'done'
  Created wheel for bs4: filename=bs4-0.0.1-py3-none-any.whl size=1265
sha256=e1fb83e5e4e646f35fd3307128ee2fa82dc925c04c2ffa82f84a495534503e6
4
  Stored in directory: c:\users\krusa\appdata\local\pip\cache\wheels\
d4\c8\5b\b5be9c20e5e4503d04a6eac8a3cd5c2393505c29f02bea0960
Successfully built bs4
Installing collected packages: bs4
Successfully installed bs4-0.0.1
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: scikit-learn in c:\users\krusa\appdata\
local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\
localcache\local-packages\python311\site-packages (1.3.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\krusa\
appdata\local\packages\
pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from scikit-learn) (1.25.2)
Requirement already satisfied: scipy>=1.5.0 in c:\users\krusa\appdata\
local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\
localcache\local-packages\python311\site-packages (from scikit-learn)
(1.11.2)
Requirement already satisfied: joblib>=1.1.1 in c:\users\krusa\
```

```
appdata\local\packages\
pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\krusa\
appdata\local\packages\
pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-
packages\python311\site-packages (from scikit-learn) (3.2.0)
Note: you may need to restart the kernel to use updated packages.
```

The provided code uses Jupyter Notebook magic commands to install several Python packages commonly used in natural language processing and machine learning tasks. These packages include NLTK for text processing, BeautifulSoup4 for web scraping and HTML/XML parsing, Contractions for handling word contractions, and scikit-learn for machine learning tasks. These libraries enable data preprocessing, web data extraction, and machine learning model development for tasks like text analysis and classification. The code ensures that the required packages are installed and ready for use in the Jupyter Notebook environment, facilitating various text-based analyses and report generation.

```
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
import re
from bs4 import BeautifulSoup
import contractions
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
from sklearn.model_selection import train_test_split

url=
"https://web.archive.org/web/20201127142707if_/https://s3.amazonaws.co
m/amazon-reviews-pds/tsv/
amazon_reviews_us_Office_Products_v1_00.tsv.gz"
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\krusa\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\krusa\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\krusa\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\krusa\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

This code imports necessary libraries and downloads NLTK resources for text preprocessing. It also sets the `url` variable to a URL pointing to an Amazon reviews dataset in TSV format. The code can be part of a data preprocessing pipeline for text analysis. However, it lacks further processing steps and the actual dataset loading. You would typically use libraries like `pandas` and `requests` or `wget` to download and load the dataset from the URL, followed by additional steps like data cleaning, tokenization, and splitting into training and testing sets using `train_test_split` for a machine learning project.

# Read Data

# Keep Reviews and Ratings

```
df = pd.read_csv('amazon_reviews_us_Office_Products_v1_00.tsv',
sep=r"\t+", usecols=['star_rating', 'review_body'])
```

```
C:\Users\krusa\AppData\Local\Temp\ipykernel_29028\3982505177.py:1:
ParserWarning: Falling back to the 'python' engine because the 'c'
engine does not support regex separators (separators > 1 char and
different from '\s+' are interpreted as regex); you can avoid this
warning by specifying engine='python'.
  df = pd.read_csv('amazon_reviews_us_Office_Products_v1_00.tsv',
sep=r"\t+", usecols=['star_rating', 'review_body'])
```

This line of code reads a TSV file named 'amazon_reviews_us_Office_Products_v1_00.tsv' into a Pandas DataFrame called 'df.' It uses a tab character as the separator and selects only the 'star_rating' and 'review_body' columns from the dataset. This allows for efficient data manipulation and analysis of these specific columns.

## We form two classes and select 50000 reviews randomly from each class.

```
df['star_rating'] = df['star_rating'].apply(lambda x: 2 if x in [4,5]
else 1)
```

This line of code modifies the 'star_rating' column in the Pandas DataFrame 'df.' It applies a lambda function to each element in the 'star_rating' column. If a value in the 'star_rating' column is either 4 or 5, it is replaced with the value 2; otherwise, it is replaced with the value 1. This operation essentially converts a rating system with 4 and 5 stars to a binary classification where 4 and 5 stars are considered positive (2) and all other ratings are considered negative (1).

```
random_state = 25
class1_samples = df[df['star_rating'] == 1].sample(n=50000,
random_state=random_state)
class2_samples = df[df['star_rating'] == 2].sample(n=50000,
random_state=random_state)
class1_samples = class1_samples.reset_index(drop=True)
class2_samples = class2_samples.reset_index(drop=True)
```

This code sets a random seed for reproducibility and selects random samples from two classes (1 and 2) in the DataFrame `df`. It creates two new DataFrames, `class1_samples` and

`class2_samples`, each containing 50,000 randomly chosen rows from their respective classes. The `reset_index(drop=True)` statements reset the index of these DataFrames for better organization.

```
final_df = pd.concat([class1_samples, class2_samples],
ignore_index=True)
final_df = final_df.reset_index(drop=True)
```

This code concatenates the two DataFrames, `class1_samples` and `class2_samples`, into a single DataFrame named `final_df`. The `ignore_index=True` argument ensures that the index is reset to maintain a continuous index for the combined DataFrame. The subsequent `reset_index(drop=True)` line further resets the index to maintain continuity. As a result, `final_df` contains the combined data with a consistent index.

# Data Cleaning

```
flenDC = final_df['review_body'].str.len().mean()
```

This line of code calculates the mean (average) length of the text in the 'review_body' column of the Pandas DataFrame 'final_df' and assigns it to the variable 'flenDC'. Essentially, it computes the average number of characters in the 'review_body' text for all the rows in the DataFrame, providing insight into the typical length of reviews in the dataset.

```
def expand_contractions(text):
    if text is None:
        return None
    return contractions.fix(str(text))

final_df['review_body'] = final_df['review_body'].str.lower()

url_pattern = r'https?://\S+|www\.\S+'
html_pattern = r'<.*?>'
non_alpha_pattern = r'[^a-zA-Z\s]'

final_df['review_body'] =
final_df['review_body'].str.replace(url_pattern, '', regex=True)
final_df['review_body'] =
final_df['review_body'].str.replace(html_pattern, '', regex=True)
final_df['review_body'] =
final_df['review_body'].str.replace(non_alpha_pattern, '', regex=True)
final_df['review_body'] = final_df['review_body'].str.replace(r'\s+',
' ', regex=True)
final_df['review_body'] =
final_df['review_body'].apply(expand_contractions)
```

This code defines a series of text preprocessing steps applied to the 'review_body' column of the DataFrame 'final_df'. Here's a breakdown of each step:

1. `expand_contractions`: This is a custom function that uses the 'contractions' library to expand contractions in text. It is applied to each element in the 'review_body' column to ensure that contractions like "can't" are converted to their full forms, such as "cannot."

2. `final_df['review_body'] = final_df['review_body'].str.lower()`: This line converts all text in the 'review_body' column to lowercase, making it consistent for further processing.

3. `url_pattern = r'https?://\S+|www\.\S+'`: This regular expression pattern (`url_pattern`) is used to identify and remove URLs and website addresses from the text.

4. `html_pattern = r'<.*?>'`: This regular expression pattern (`html_pattern`) identifies and removes HTML tags from the text, effectively cleaning any HTML markup.

5. `non_alpha_pattern = r'[^a-zA-Z\s]'`: This regular expression pattern (`non_alpha_pattern`) matches any characters that are not alphabetic letters or whitespace. It is used to remove non-alphabetic characters from the text.

6. The next several lines apply the defined regular expression patterns to the 'review_body' column using the `str.replace()` method, effectively removing URLs, HTML tags, non-alphabetic characters, and extra whitespace.

After running these preprocessing steps, the 'review_body' column in the 'final_df' DataFrame will be cleaned and ready for further text analysis or machine learning tasks.

```
slenDC = final_df['review_body'].str.len().mean()
print(f"{flenDC},{slenDC}")
```

```
185.14938,185.19607
```

# Pre-processing

## remove the stop words

## perform lemmatization

```
flenPP = final_df['review_body'].str.len().mean()

lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    if text is None:
        return None
```

```
    tokens = nltk.word_tokenize(text)
    filtered_tokens = [word for word in tokens if word not in
stop_words]
    lemmatized_tokens = [lemmatizer.lemmatize(word) for word in
filtered_tokens]
    return ' '.join(lemmatized_tokens)

final_df['review_body'] =
final_df['review_body'].apply(preprocess_text)
```

This code defines a text preprocessing function `preprocess_text` and applies it to the 'review_body' column of the 'final_df' DataFrame. Here's what each part of the code does:

1. `lemmatizer = WordNetLemmatizer()`: It initializes a WordNet lemmatizer, which is used for lemmatization, reducing words to their base or dictionary form.
2. `stop_words = set(stopwords.words('english'))`: It creates a set of English stopwords using NLTK's built-in stopwords list. These stopwords are words that are commonly removed from text during preprocessing because they typically do not provide meaningful information.
3. `def preprocess_text(text)`: This is a custom preprocessing function that takes an input text as an argument.
    - It tokenizes the text into individual words using `nltk.word_tokenize(text)`.
    - Then, it removes any stopwords from the tokenized words, ensuring that common, non-informative words are excluded.
    - Next, it lemmatizes each word using the WordNet lemmatizer, converting words to their base forms.
    - Finally, it returns the preprocessed text as a string by joining the lemmatized tokens with spaces.
4. `final_df['review_body'] = final_df['review_body'].apply(preprocess_text)`: This line applies the `preprocess_text` function to each element in the 'review_body' column of the 'final_df' DataFrame. It effectively preprocesses the text in the 'review_body' column by tokenizing, removing stopwords, and lemmatizing the words.

After executing this code, the 'review_body' column in 'final_df' will contain preprocessed text data ready for use in natural language processing or machine learning tasks.

```
slenPP = final_df['review_body'].str.len().mean()
print(f"{flenPP},{slenPP}")

185.19607,185.04318
```

## Split Data into two parts (train_df and test_df)

```
train_df, test_df = train_test_split(final_df, test_size=0.2,
random_state=random_state)

train_df = train_df.reset_index(drop=True)
test_df = test_df.reset_index(drop=True)

train_labels = train_df['star_rating']
test_labels = test_df['star_rating']
```

The provided code performs data splitting for machine learning tasks. It begins by using scikit-learn's train_test_split function to divide the 'final_df' DataFrame into two sets: 'train_df' (80% of the data) and 'test_df' (20% of the data). The test_size parameter controls the portion allocated for testing, while random_state ensures the process is reproducible.

Subsequently, both 'train_df' and 'test_df' have their indices reset to create a continuous and consistent index structure.

Finally, the code extracts the 'star_rating' column from both 'train_df' and 'test_df' and assigns them to 'train_labels' and 'test_labels,' respectively. These labels are essential for supervised machine learning, where the goal is often to predict or classify based on the provided labels.

# TF-IDF and BoW Feature Extraction

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(max_features=10000)

tfidf_train_features =
tfidf_vectorizer.fit_transform(train_df['review_body'])
tfidf_test_features =
tfidf_vectorizer.transform(test_df['review_body'])
```

This code utilizes scikit-learn's `TfidfVectorizer` to convert text data into TF-IDF features. It limits the number of features to a maximum of 10,000. The code transforms the text in the 'review_body' column of the training dataset ('train_df') into TF-IDF features stored in 'tfidf_train_features'. Similarly, it transforms the text in the testing dataset ('test_df') into TF-IDF features stored in 'tfidf_test_features'. TF-IDF quantifies word importance in text, aiding machine learning tasks, and the 10,000-feature limit helps manage dimensionality for efficient modeling.

```
from sklearn.feature_extraction.text import CountVectorizer

bow_vectorizer = CountVectorizer(max_features=10000)

bow_train_features =
```

```
bow_vectorizer.fit_transform(train_df['review_body'])
bow_test_features = bow_vectorizer.transform(test_df['review_body'])
```

The code imports `CountVectorizer`, initializes it with a maximum of 10,000 features, and then applies it to the 'review_body' column of both the training ('train_df') and testing ('test_df') datasets. This vectorizer transforms the text data into numerical representations based on word frequency. 'bow_train_features' and 'bow_test_features' store the resulting BoW representations for the training and testing data, respectively. BoW represents each document as a vector of word frequencies, making it suitable for various text analysis tasks, including machine learning.

# Perceptron Using Both Features

```
from sklearn.linear_model import Perceptron
from sklearn.metrics import precision_score, recall_score, f1_score
```

# 1. Perceptron for TF-IDF

```
perceptron_tfidf = Perceptron()

perceptron_tfidf.fit(tfidf_train_features, train_labels)

tfidf_predictions = perceptron_tfidf.predict(tfidf_test_features)

precision_tfidf = precision_score(test_labels, tfidf_predictions)
recall_tfidf = recall_score(test_labels, tfidf_predictions)
f1_tfidf = f1_score(test_labels, tfidf_predictions)

print(f'{precision_tfidf:.4f} {recall_tfidf:.4f} {f1_tfidf:.4f}')

0.7756 0.8279 0.8009
```

1. `perceptron_tfidf = Perceptron()`: It initializes a Perceptron classifier.
2. `perceptron_tfidf.fit(tfidf_train_features, train_labels)`: The Perceptron is trained using TF-IDF features (`tfidf_train_features`) and corresponding training labels (`train_labels`).
3. `tfidf_predictions = perceptron_tfidf.predict(tfidf_test_features)`: The trained Perceptron predicts labels for the testing dataset (`tfidf_test_features`), storing the results in `tfidf_predictions`.
4. Precision, recall, and F1-score are computed using scikit-learn's `precision_score`, `recall_score`, and `f1_score` functions.

5. The calculated precision, recall, and F1-score are printed, representing the classifier's performance on the test data. These metrics help evaluate the model's accuracy, completeness, and overall effectiveness in classification.

# 2. Perceptron for BoW

```
perceptron_bow = Perceptron()

perceptron_bow.fit(bow_train_features, train_labels)

bow_predictions = perceptron_bow.predict(bow_test_features)

precision_bow = precision_score(test_labels, bow_predictions)
recall_bow = recall_score(test_labels, bow_predictions)
f1_bow = f1_score(test_labels, bow_predictions)

print(f'{precision_bow:.4f} {recall_bow:.4f} {f1_bow:.4f}')
```

```
0.7996 0.7862 0.7928
```

# SVM Using Both Features

# 1. SVM for TF-IDF

```
from sklearn.svm import LinearSVC
from sklearn.metrics import precision_score, recall_score, f1_score

svm_tfidf = LinearSVC(dual=True, random_state=random_state)

svm_tfidf.fit(tfidf_train_features, train_labels)

svm_tfidf_predictions = svm_tfidf.predict(tfidf_test_features)

precision_tfidf = precision_score(test_labels, svm_tfidf_predictions)
recall_tfidf = recall_score(test_labels, svm_tfidf_predictions)
f1_tfidf = f1_score(test_labels, svm_tfidf_predictions)

print(f'{precision_tfidf:.4f} {recall_tfidf:.4f} {f1_tfidf:.4f}')
```

```
0.8402 0.8455 0.8428
```

1. `from sklearn.svm import LinearSVC`: Import the LinearSVC classifier from scikit-learn.
2. `from sklearn.metrics import precision_score, recall_score, f1_score`: Import functions for precision, recall, and F1-score calculations.

3. `svm_tfidf = LinearSVC(dual=True, random_state=random_state)`: Initialize a LinearSVC classifier with specified options, including `dual=True` and a random seed for reproducibility.

4. `svm_tfidf.fit(tfidf_train_features, train_labels)`: Train the LinearSVC classifier using the TF-IDF features and training labels.

5. `svm_tfidf_predictions = svm_tfidf.predict(tfidf_test_features)`: Predict labels for the test dataset using the trained model.

# 2. SVM for BoW

```
from sklearn.svm import LinearSVC
from sklearn.metrics import precision_score, recall_score, f1_score

svm_bow = LinearSVC(max_iter=10000, random_state=42)

svm_bow.fit(bow_train_features, train_labels)

svm_bow_predictions = svm_bow.predict(bow_test_features)

precision_bow = precision_score(test_labels, svm_bow_predictions)
recall_bow = recall_score(test_labels, svm_bow_predictions)
f1_bow = f1_score(test_labels, svm_bow_predictions)

print(f'{precision_bow:.4f} {recall_bow:.4f} {f1_bow:.4f}')
```

```
C:\Users\krusa\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages\sklearn\svm\_classes.py:32:
FutureWarning: The default value of `dual` will change from `True` to
``'auto'`` in 1.5. Set the value of `dual` explicitly to suppress the
warning.
  warnings.warn(

0.8404 0.8197 0.8299
```

# Logistic Regression Using Both Features

# 1. LR - TFIDF

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, f1_score

logistic_regression_tfidf = LogisticRegression(max_iter=10000,
```

```
random_state=42)

logistic_regression_tfidf.fit(tfidf_train_features, train_labels)

logistic_regression_tfidf_predictions =
logistic_regression_tfidf.predict(tfidf_test_features)

precision_tfidf = precision_score(test_labels,
logistic_regression_tfidf_predictions)
recall_tfidf = recall_score(test_labels,
logistic_regression_tfidf_predictions)
f1_tfidf = f1_score(test_labels,
logistic_regression_tfidf_predictions)

print(f'{precision_tfidf:.4f} {recall_tfidf:.4f} {f1_tfidf:.4f}')
```

```
0.8421 0.8573 0.8497
```

1. `from sklearn.linear_model import LogisticRegression`: Import the LogisticRegression classifier from scikit-learn.
2. `from sklearn.metrics import precision_score, recall_score, f1_score`: Import functions for precision, recall, and F1-score calculations.
3. `logistic_regression_tfidf = LogisticRegression(max_iter=10000, random_state=42)`: Initialize a Logistic Regression classifier with specified options, including a high maximum number of iterations (`max_iter=10000`) and a random seed for reproducibility.
4. `logistic_regression_tfidf.fit(tfidf_train_features, train_labels)`: Train the Logistic Regression classifier using the TF-IDF features and training labels.
5. `logistic_regression_tfidf_predictions = logistic_regression_tfidf.predict(tfidf_test_features)`: Predict labels for the test dataset using the trained model.

# 2. LR - BoW

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, f1_score

logistic_regression_bow = LogisticRegression(max_iter=10000,
random_state=42)

logistic_regression_bow.fit(bow_train_features, train_labels)

logistic_regression_bow_predictions =
```

```
logistic_regression_bow.predict(bow_test_features)

precision_bow = precision_score(test_labels,
logistic_regression_bow_predictions)
recall_bow = recall_score(test_labels,
logistic_regression_bow_predictions)
f1_bow = f1_score(test_labels, logistic_regression_bow_predictions)

print(f'{precision_bow:.4f} {recall_bow:.4f} {f1_bow:.4f}')

0.8488 0.8320 0.8403
```

# Naive Bayes Using Both Features

# 1. Naive Bayes Tf-IDF

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import precision_score, recall_score, f1_score

naive_bayes_tfidf = MultinomialNB()

naive_bayes_tfidf.fit(tfidf_train_features, train_labels)

naive_bayes_tfidf_predictions =
naive_bayes_tfidf.predict(tfidf_test_features)

precision_tfidf = precision_score(test_labels,
naive_bayes_tfidf_predictions)
recall_tfidf = recall_score(test_labels,
naive_bayes_tfidf_predictions)
f1_tfidf = f1_score(test_labels, naive_bayes_tfidf_predictions)

print(f'{precision_tfidf:.4f} {recall_tfidf:.4f} {f1_tfidf:.4f}')

0.8289 0.8111 0.8199
```

1. `from sklearn.naive_bayes import MultinomialNB`: Import the Multinomial Naive Bayes classifier from scikit-learn.
2. `from sklearn.metrics import precision_score, recall_score, f1_score`: Import functions for precision, recall, and F1-score calculations.
3. `naive_bayes_tfidf = MultinomialNB()`: Initialize a Multinomial Naive Bayes classifier.
4. `naive_bayes_tfidf.fit(tfidf_train_features, train_labels)`: Train the Multinomial Naive Bayes classifier using the TF-IDF features and training labels.

5. `naive_bayes_tfidf_predictions = naive_bayes_tfidf.predict(tfidf_test_features)`: Predict labels for the test dataset using the trained model.

# 2. Naive Bayes BoW

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import precision_score, recall_score, f1_score

naive_bayes_bow = MultinomialNB()

naive_bayes_bow.fit(bow_train_features, train_labels)

naive_bayes_bow_predictions =
naive_bayes_bow.predict(bow_test_features)

precision_bow = precision_score(test_labels,
naive_bayes_bow_predictions)
recall_bow = recall_score(test_labels, naive_bayes_bow_predictions)
f1_bow = f1_score(test_labels, naive_bayes_bow_predictions)

print(f'{precision_bow:.4f} {recall_bow:.4f} {f1_bow:.4f}')

0.8455 0.7489 0.7943
```