Dependencies

Install

```
!pip install contractions
!pip install ipython-autotime
```

```
Collecting contractions
  Downloading contractions-0.1.73-py2.py3-none-any.whl (8.7 kB)
Collecting textsearch>=0.0.21 (from contractions)
  Downloading textsearch-0.0.24-py2.py3-none-any.whl (7.6 kB)
Collecting anyascii (from textsearch>=0.0.21->contractions)
  Downloading anyascii-0.3.2-py3-none-any.whl (289 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 289.9/289.9 kB 3.9 MB/s eta 0:00:00
Collecting pyahocorasick (from textsearch>=0.0.21->contractions)
  Downloading pyahocorasick-2.0.0-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.whl (110 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 110.8/110.8 kB 5.1 MB/s eta 0:00:00
Installing collected packages: pyahocorasick, anyascii, textsearch, contractions
Successfully installed anyascii-0.3.2 contractions-0.1.73 pyahocorasick-2.0.0 textsearch-0.0.24
Collecting ipython-autotime
  Downloading ipython_autotime-0.3.1-py2.py3-none-any.whl (6.8 kB)
Requirement already satisfied: ipython in /usr/local/lib/python3.10/dist-packages (from ipython-autotime) (7.34.0)
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.10/dist-packages (from ipython->ipython-autotime) (67.7.2)
Collecting jedi>=0.16 (from ipython->ipython-autotime)
  Downloading jedi-0.19.0-py2.py3-none-any.whl (1.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 17.8 MB/s eta 0:00:00
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-packages (from ipython->ipython-autotime) (4.4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-packages (from ipython->ipython-autotime) (0.7.5)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.10/dist-packages (from ipython->ipython-autotime) (5.7.1)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from ipython->ipython-autotime) (3.0.39)
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (from ipython->ipython-autotime) (2.16.1)
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-packages (from ipython->ipython-autotime) (0.2.0)
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.10/dist-packages (from ipython->ipython-autotime) (0.1.6)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-packages (from ipython->ipython-autotime) (4.8.0)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in /usr/local/lib/python3.10/dist-packages (from jedi>=0.16->ipython->ipython-autotime) (0.8.3)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.10/dist-packages (from pexpect>4.3->ipython->ipython-autotime) (0.7.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-packages (from prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->ipython->ipython-autotime) (0.2.6)
Installing collected packages: jedi, ipython-autotime
Successfully installed ipython-autotime-0.3.1 jedi-0.19.0
```

Imports

```
import os
import re
import unicodedata

import warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd

import nltk
from nltk.corpus import stopwords, wordnet
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize

nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')

import contractions

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.metrics import precision_score, recall_score, f1_score

from sklearn.linear_model import Perceptron, LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.naive_bayes import MultinomialNB

from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingGridSearchCV, GridSearchCV

%load_ext autotime
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
time: 602 µs (started: 2023-09-18 04:58:29 +00:00)
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
```

Dataset Preparation

```
class Config:
    RANDOM_STATE = 56
    DATA_PATH = "amazon_reviews_us_Office_Products_v1_00.tsv.gz"
    TEST_SPLIT = 0.2
    N_SAMPLES_EACH_CLASS = 50000
    NUM_TFIDF_FEATURES = 5000
    NUM_BOW_FEATURES = 8000
```

```
time: 903 µs (started: 2023-09-18 04:58:29 +00:00)
```

Download Data

```
# %%bash
# cd "/content/drive/MyDrive/Colab Notebooks/CSCI544/HW1"
# curl -o amazon_reviews_us_Office_Products_v1_00.tsv.gz \
# https://web.archive.org/web/20201127142707if_/https://s3.amazonaws.com/amazon-reviews-pds/tsv/amazon_reviews_us_Office_Products_v1_00.tsv.gz
```

```
time: 422 µs (started: 2023-09-18 04:58:29 +00:00)
```

Read Data

- `sep='\t'` : Values in the TSV file are separated by tabs
- `on_bad_lines="skip"` : Skip any lines in the file that are improperly formatted or contain errors.
- `memory_map=True` : Maps the file obj directly to memory for direct access improving performance for large files
- `nrows=10` : Limits the number of rows to read from the file to 10.
- `usecols=["review_headline","review_body", "star_rating"]` : Only select subset of columns to read - faster parsing time and low memory usage.

```
os.chdir("/content/drive/MyDrive/Colab Notebooks/CSCI544/HW1")
```

```
time: 197 ms (started: 2023-09-18 04:58:29 +00:00)
```

Have quick look at dataset by reading 10 rows to get the column names

```
df_small = pd.read_csv(
    Config.DATA_PATH,
    sep='\t',
    on_bad_lines="skip",
    memory_map=True,
    nrows=10
)
df_small.columns
```

```
Index(['marketplace', 'customer_id', 'review_id', 'product_id',
       'product_parent', 'product_title', 'product_category', 'star_rating',
       'helpful_votes', 'total_votes', 'vine', 'verified_purchase',
       'review_headline', 'review_body', 'review_date'],
      dtype='object')time: 831 ms (started: 2023-09-18 04:58:30 +00:00)
```

```
del df_small
```

```
time: 661 µs (started: 2023-09-18 04:58:30 +00:00)
```

Read the entire data

```
df = pd.read_csv(
    Config.DATA_PATH,
    sep='\t',
    usecols=["review_headline","review_body", "star_rating"],
    on_bad_lines="skip",
    memory_map=True,
)
```

```
time: 49.9 s (started: 2023-09-18 04:58:30 +00:00)
```

```
df.head()
```

	star_rating	review_headline	review_body
0	5	Five Stars	Great product.
1	5	Phffffffft, Phfffffft. Lots of air, and it's C...	What's to say about this commodity item except...
2	5	but I am sure I will like it.	Haven't used yet, but I am sure I will like it.
3	1	and the shredder was dirty and the bin was par...	Although this was labeled as "new" the...

Keep Reviews and Ratings

```
# Select columns by name
df_filtered = df.loc[:,['review_body', 'star_rating']]
df_filtered.head()
```

	review_body	star_rating
0	Great product.	5
1	What's to say about this commodity item except...	5
2	Haven't used yet, but I am sure I will like it.	5
3	Although this was labeled as "new" the...	1
4	Gorgeous colors and easy to use	4

```
time: 139 ms (started: 2023-09-18 04:59:20 +00:00)
```

Create Binary Classification Problem

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

Handling the inconsistencies `star_rating` columns:

- **Converting 'star_rating' to Numeric:**
  - The 'star_rating' column likely contains numerical values, but they might be stored as strings or in a format which can cause issues for further analysis or modeling.
  - Convert the column to numeric, and replace non-convertible with NaN.

2. **Handling Missing Values:**
   - After converting to numeric, there might be rows with missing or non-convertible values, which are now represented as NaN.
   - Drop the rows with NaN values
3. **Classification of Ratings:**
   - The task requires binary classification based on the ratings, where ratings 1, 2, and 3 form one class (class 1), and ratings 4 and 5 form another class (class 2).
   - We apply the mapping as per requirements.

```
# Check incosistencies in star_rating column
df['star_rating'].unique()
```

```
array([5, 1, 4, 2, 3, '5', '1', '3', '4', '2', '2015-06-05', '2015-02-11',
       nan, '2014-02-14'], dtype=object)time: 175 ms (started: 2023-09-18 04:59:20 +00:00)
```

```python
# Convert the 'star_rating' column to numeric, coerce errors to NaN
df_filtered['star_rating'] = pd.to_numeric(df['star_rating'], errors='coerce')

# Drop NaN values from `star_rating`
df_filtered.dropna(subset=["star_rating"], inplace=True)

# Classify ratings as 1, 2, or 3 into class 1, and ratings 4 and 5 into class 2
df_filtered['sentiment'] = df_filtered['star_rating'].apply(
    lambda x: 1 if x <= 3 else 2
)

print("Shape of unfiltered dataframe:", df.shape)
print("Shape of filtered dataframe:", df_filtered.shape)

df_filtered.head()
```

```
Shape of unfiltered dataframe: (2640352, 3)
Shape of filtered dataframe: (2640335, 3)
```

	review_body	star_rating	sentiment
0	Great product.	5.0	2
1	What's to say about this commodity item except...	5.0	2
2	Haven't used yet, but I am sure I will like it.	5.0	2
3	Although this was labeled as "new" the...	1.0	1
4	Gorgeous colors and easy to use	4.0	2

```
time: 2.38 s (started: 2023-09-18 04:59:21 +00:00)
```

Sampling data

- Find indices of each class
- Choose random 50000 values using `sample` function for each class
- Resample for shuffling

```python
# Create a new DataFrame with sampled data
balanced_df = pd.concat(
    [
        df_filtered.query('sentiment==1').sample(
            n=Config.N_SAMPLES_EACH_CLASS, random_state=Config.RANDOM_STATE
        ),
        df_filtered.query('sentiment==2').sample(
            n=Config.N_SAMPLES_EACH_CLASS, random_state=Config.RANDOM_STATE
        )
    ],
    ignore_index=True
).sample(frac=1, random_state=Config.RANDOM_STATE)

balanced_df.drop(columns=["star_rating"], inplace=True)

# Handling non-string values in Reviews
balanced_df["review_body"] = balanced_df["review_body"].astype(str)

balanced_df.head()
```

	review_body	sentiment
7527	Agree with other posters in that these worked ...	1
84247	These are a little smaller than the ones I had...	2
79106	Well made, plenty of card pockets, and I like ...	2
37339	In the same year, I bought a new HP laptop and...	1
33018	I bought this on Jan 2011 as a gift. I turned ...	1

```
time: 464 ms (started: 2023-09-18 04:59:23 +00:00)
```

Data Cleaning

- Using regex expressions to match and replace the below with with empty strings
  - emails
  - URLs
  - HTML tags
  - punctautions
  - extra spaces
  - non-alphabetical characters
- We use contractions to expand contractions like `I'll` to `I will`
  - This also reduces the of words in the vocabulary
  - The expanded forms of contraction may or may not fit into the context of sentence

> Vectorize the `clean_text` function for better performance

```python
def unicode_to_ascii(s):
  return ''.join(c for c in unicodedata.normalize('NFD', s)
      if unicodedata.category(c) != 'Mn')

def expand_contractions(text):
    return contractions.fix(text)

def clean_text(text):
    text = unicode_to_ascii(text.lower().strip())

    # replacing email addresses with empty string
    text = re.sub(
        r"[a-zA-Z0-9_\-\.]+@[a-zA-Z0-9_\-\.]+\.[a-zA-Z]{2,5}", " ", text
    )

    # replacing urls with empty string
    text = re.sub(
        r"\bhttps?:\/\/\S+|www\.\S+", " ", text
    )

    # Remove HTML tags with empty string
    text = re.sub(r"<.*?>", "", text)
```

```
    # Expand contraction for eg., wouldn't => would not
    text = expand_contractions(text)

    # creating a space between a word and the punctuation following it
    text = re.sub(r"([?.!,¿])", r" \1 ", text)
    text = re.sub(r'[" "]+', " ", text)

    # removes all non-alphabetical characters
    text = re.sub(r"[^a-zA-Z\s]+", "", text)

    # remove extra spaces
    text = re.sub(" +", " ", text)

    text = text.strip()
    return text

clean_text_vect = np.vectorize(clean_text)
```

```
time: 965 µs (started: 2023-09-18 04:59:24 +00:00)
```

```
# Calculate average length of reviews before cleaning
avg_len_before_clean = balanced_df["review_body"].apply(len).mean()

balanced_df["review_body"] = balanced_df["review_body"].apply(clean_text_vect)

# Drop rows with empty review_body
balanced_df = balanced_df.loc[balanced_df["review_body"].str.strip() != ""]

# Calculate average length of reviews after cleaning
avg_len_after_clean = balanced_df["review_body"].apply(len).mean()

print(f'Avg. Length of Reviews Before Cleaning: {avg_len_before_clean:.2f} characters')
print(f'Avg. Length of Reviews After Cleaning: {avg_len_after_clean:.2f} characters')
```

```
Avg. Length of Reviews Before Cleaning: 314.91 characters
Avg. Length of Reviews After Cleaning: 299.82 characters
time: 50.5 s (started: 2023-09-18 04:59:24 +00:00)
```

Pre-processing

- Remove the stopwords
  - Do not exclude negative stopwords
- Lemmatize words after tokenization
  - with Pos Tagging
    - Boosts the precision and comprehensibility of textual information by converting words to their fundamental forms
    - Considers their grammatical functions within sentences
  - without Pos Tagging

> Vectorize the `preprocess_text` function for better performance

```
# Stopword list
og_stopwords = set(stopwords.words('english'))
# Define a list of negative words to remove
neg_words = ['no', 'not', 'nor', 'neither', 'none', 'never', 'nobody', 'nowhere']
custom_stopwords = [word for word in og_stopwords if word not in neg_words]

pattern = re.compile(r'\b('+r'|'.join(custom_stopwords)+r')\b\s*')

lemmatizer = WordNetLemmatizer()

def pos_tagger(nltk_tag):
    if nltk_tag.startswith('J'):
        return wordnet.ADJ
    elif nltk_tag.startswith('V'):
        return wordnet.VERB
    elif nltk_tag.startswith('N'):
        return wordnet.NOUN
    elif nltk_tag.startswith('R'):
        return wordnet.ADV
    else:
        return None

def lemmatize_text_with_pos_tagging(text):
    words = nltk.pos_tag(word_tokenize(text))
    words = map(lambda x: (x[0], pos_tagger(x[1])), words)
    lemmatized_words = [
        lemmatizer.lemmatize(word, tag)
        if tag else word
        for word, tag in words
    ]
    return ' '.join(lemmatized_words)

def lemmatize_text(text):
    words = word_tokenize(text)
    lemmatized_words = [lemmatizer.lemmatize(word) for word in words]
    return ' '.join(lemmatized_words)

def preprocess_text(text):
    # replacing all the stopwords
    text = pattern.sub('',text)
    text = lemmatize_text(text)
    return text

def preprocess_text_with_pos_tagging(text):
    # replacing all the stopwords
    text = pattern.sub('',text)
    text = lemmatize_text_with_pos_tagging(text)
    return text

preprocess_text_vect = np.vectorize(preprocess_text)
preprocess_text_vect_pos_tag = np.vectorize(preprocess_text_with_pos_tagging)
```

```
time: 4.02 ms (started: 2023-09-18 05:24:11 +00:00)
```

Without Pos Tag

```
# Calculate average length of reviews before cleaning
avg_len_before_preprocess = avg_len_after_clean

```

```
balanced_df["review_body"] = balanced_df["review_body"].apply(preprocess_text_vect)

# Drop rows with empty review_body
balanced_df = balanced_df.loc[balanced_df["review_body"].str.strip() != ""]

# Calculate average length of reviews after cleaning
avg_len_after_preprocess = balanced_df["review_body"].apply(len).mean()

print(f'Avg. Length of Reviews Before Preprocessing: {avg_len_before_preprocess:.2f} characters')
print(f'Avg. Length of Reviews After Preprocessing: {avg_len_after_preprocess:.2f} characters')
```

```
Avg. Length of Reviews Before Preprocessing: 299.82 characters
Avg. Length of Reviews After Preprocessing: 189.75 characters
time: 1min 34s (started: 2023-09-18 05:00:14 +00:00)
```

With Pos Tag

```
balanced_df_pos_tag = balanced_df.copy(deep=True)
balanced_df_pos_tag["review_body"] = balanced_df_pos_tag["review_body"].apply(preprocess_text_vect_pos_tag)

# Drop rows with empty review_body
balanced_df_pos_tag = balanced_df_pos_tag.loc[balanced_df["review_body"].str.strip() != ""]

# Calculate average length of reviews after cleaning
avg_len_after_preprocess_pos_tag = balanced_df_pos_tag["review_body"].apply(len).mean()

print(f'Avg. Length of Reviews Before Preprocessing: {avg_len_before_preprocess:.2f} characters')
print(f'Avg. Length of Reviews After Preprocessing: {avg_len_after_preprocess_pos_tag:.2f} characters')
```

```
Avg. Length of Reviews Before Preprocessing: 299.82 characters
Avg. Length of Reviews After Preprocessing: 182.71 characters
time: 10min 35s (started: 2023-09-18 05:27:15 +00:00)
```

Train and Test Split

```
# Without Pos Tagging lem
X_train, X_test, y_train, y_test = train_test_split(
    balanced_df['review_body'],
    balanced_df['sentiment'],
    test_size=Config.TEST_SPLIT,
    random_state=Config.RANDOM_STATE
)

# With Pos tagging lem
X_train_pt, X_test_pt, y_train_pt, y_test_pt = train_test_split(
    balanced_df_pos_tag['review_body'],
    balanced_df_pos_tag['sentiment'],
    test_size=Config.TEST_SPLIT,
    random_state=Config.RANDOM_STATE
)
```

```
time: 51.1 ms (started: 2023-09-18 05:39:02 +00:00)
```

Feature Extraction

TF-IDF

```
# Without Pos Tagging lem
tfidf_vectorizer = TfidfVectorizer(max_features=Config.NUM_TFIDF_FEATURES)
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

# With Pos tagging lem
tfidf_vectorizer_pt = TfidfVectorizer(max_features=Config.NUM_TFIDF_FEATURES)
X_train_tfidf_pt = tfidf_vectorizer_pt.fit_transform(X_train_pt)
X_test_tfidf_pt = tfidf_vectorizer_pt.transform(X_test_pt)
```

```
time: 16.9 s (started: 2023-09-18 05:39:36 +00:00)
```

Bag of words

```
count_vectorizer = CountVectorizer(max_features=Config.NUM_BOW_FEATURES)
X_train_bow = count_vectorizer.fit_transform(X_train)
X_test_bow = count_vectorizer.transform(X_test)

count_vectorizer_pt = CountVectorizer(max_features=Config.NUM_BOW_FEATURES)
X_train_bow_pt = count_vectorizer_pt.fit_transform(X_train_pt)
X_test_bow_pt = count_vectorizer_pt.transform(X_test_pt)
```

```
time: 5.99 s (started: 2023-09-18 05:39:56 +00:00)
```

ML Modeling

```
def evaluate_model(model, X_test, y_test):
    # Predict on the test set
    y_pred = model.predict(X_test)

    # Calculate evaluation metrics
    precision = precision_score(y_test, y_pred, average='binary')
    recall = recall_score(y_test, y_pred, average='binary')
    f1 = f1_score(y_test, y_pred, average='binary')

    return precision, recall, f1


def train_and_evaluate_model(model_class, X_train, y_train, X_test, y_test, **model_params):
    # Initialize model
    model = model_class(**model_params)

    # Train the model
    model.fit(X_train, y_train)

    # Evaluate model
```

```
    precision, recall, f1 = evaluate_model(model, X_test, y_test)
```

```
time: 1.31 ms (started: 2023-09-18 05:40:02 +00:00)
```

Perceptron Using Both Features

Grid Search

Performed Grid search changing hyperparameters

- `max_iter` - number of epochs
- `penalty` - regularization function
- `tol` - loss to stop the iteration

```
# Define the parameter grid to search
param_grid = {
    'max_iter': [1000, 2000, 4000, 8000, 10000, 12000],
    'tol': [1e-1, 1e-2, 1e-3, 1e-4, 1e-5],
    'penalty': ['l2','l1','elasticnet']
}

# Initialize Perceptron model
perceptron = Perceptron()

# Grid search for BoW features
grid_search_bow = GridSearchCV(
    estimator=perceptron,
    param_grid=param_grid,
    scoring='f1',
    cv=5,  # Number of cross-validation folds
    n_jobs=-1,
    verbose=2
)

grid_search_bow.fit(X_train_bow, y_train)

# Get the best parameters and model for BoW
best_params_bow = grid_search_bow.best_params_
best_model_bow = grid_search_bow.best_estimator_

# Evaluate the best model for BoW
precision_perceptron_bow, recall_perceptron_bow, f1_perceptron_bow = evaluate_model(best_model_bow, X_test_bow, y_test)

# Print the results for BoW
print(f'Best Parameters (BoW): {best_params_bow}')
print(f'Precision Recall F1 (Perceptron, BoW): {precision_perceptron_bow:.4f} {recall_perceptron_bow:.4f} {f1_perceptron_bow:.4f}')
```

```
Fitting 5 folds for each of 90 candidates, totalling 450 fits
Best Parameters (BoW): {'max_iter': 1000, 'penalty': 'l1', 'tol': 0.01}
Precision Recall F1 (Perceptron, BoW): 0.8521 0.7558 0.8011
time: 3min 55s (started: 2023-09-18 05:44:36 +00:00)
```

```
# Grid search for TF-IDF features
grid_search_tfidf = GridSearchCV(
    estimator=perceptron,
    param_grid=param_grid,
    scoring='f1',
    cv=5,  # Number of cross-validation folds
    n_jobs=-1,
    verbose=4
)

grid_search_tfidf.fit(X_train_tfidf, y_train)

# Get the best parameters and model for TF-IDF
best_params_tfidf = grid_search_tfidf.best_params_
best_model_tfidf = grid_search_tfidf.best_estimator_

# Evaluate the best model for TF-IDF
precision_perceptron_tfidf, recall_perceptron_tfidf, f1_perceptron_tfidf = evaluate_model(best_model_tfidf, X_test_tfidf, y_test)

# Print the results for TF-IDF
print(f'Best Parameters (TF-IDF): {best_params_tfidf}')
print(f'Precision Recall F1 (Perceptron, TF-IDF): {precision_perceptron_tfidf:.4f} {recall_perceptron_tfidf:.4f} {f1_perceptron_tfidf:.4f}')
```

```
Fitting 5 folds for each of 90 candidates, totalling 450 fits
Best Parameters (TF-IDF): {'max_iter': 1000, 'penalty': 'elasticnet', 'tol': 0.01}
Precision Recall F1 (Perceptron, TF-IDF): 0.8440 0.7018 0.7663
time: 1min 56s (started: 2023-09-18 05:55:19 +00:00)
```

Without Pos Tag Lem

```
# Train and evaluate Perceptron model using BoW features
precision_perceptron_bow, recall_perceptron_bow, f1_perceptron_bow = train_and_evaluate_model(
    Perceptron,
    X_train_bow, y_train, X_test_bow, y_test,
    max_iter=1000
)

# Train and evaluate Perceptron model using TF-IDF features
precision_perceptron_tfidf, recall_perceptron_tfidf, f1_perceptron_tfidf = train_and_evaluate_model(
    Perceptron,
    X_train_tfidf, y_train, X_test_tfidf, y_test,
    max_iter=1000
)

# Print the results
print(f'Precision Recall F1 (BoW): {precision_perceptron_bow:.4f} {recall_perceptron_bow:.4f} {f1_perceptron_bow:.4f}')
print(f'Precision Recall F1 (TF-IDF): {precision_perceptron_tfidf:.4f} {recall_perceptron_tfidf:.4f} {f1_perceptron_tfidf:.4f}')
```

```
Precision Recall F1 (BoW): 0.7989 0.8290 0.8137
Precision Recall F1 (TF-IDF): 0.8213 0.8153 0.8183
time: 2.15 s (started: 2023-09-18 05:54:12 +00:00)
```

With Pos Tag Lem

```
# Train and evaluate Perceptron model using BoW features
precision_perceptron_bow_pt, recall_perceptron_bow_pt, f1_perceptron_bow_pt = train_and_evaluate_model(
    Perceptron,
    X_train_bow_pt, y_train, X_test_bow_pt, y_test,
    max_iter=1000
```

```
)

# Train and evaluate Perceptron model using TF-IDF features
precision_perceptron_tfidf_pt, recall_perceptron_tfidf_pt, f1_perceptron_tfidf_pt = train_and_evaluate_model(
    Perceptron,
    X_train_tfidf_pt, y_train, X_test_tfidf_pt, y_test,
    max_iter=1000
)

# Print the results
print(f'Precision Recall F1 (BoW): {precision_perceptron_bow_pt:.4f} {recall_perceptron_bow_pt:.4f} {f1_perceptron_bow_pt:.4f}')
print(f'Precision Recall F1 (TF-IDF): {precision_perceptron_tfidf_pt:.4f} {recall_perceptron_tfidf_pt:.4f} {f1_perceptron_tfidf_pt:.4f}')
```

```
Precision Recall F1 (BoW): 0.8412 0.7556 0.7961
Precision Recall F1 (TF-IDF): 0.8053 0.8120 0.8087
time: 1.17 s (started: 2023-09-18 05:59:15 +00:00)
```

SVM Using Both Features

Without Pos Tag Lem

```
# Train and evaluate SVM model using BoW features
precision_svm_bow, recall_svm_bow, f1_svm_bow = train_and_evaluate_model(
    LinearSVC,
    X_train_bow, y_train, X_test_bow, y_test,
    max_iter=2000
)

# Train and evaluate SVM model using TF-IDF features
precision_svm_tfidf, recall_svm_tfidf, f1_svm_tfidf = train_and_evaluate_model(
    LinearSVC,
    X_train_tfidf, y_train, X_test_tfidf, y_test,
    max_iter=2000
)

# Print the results
print(f'Precision Recall F1 (SVM, BoW): {precision_svm_bow:.4f} {recall_svm_bow:.4f} {f1_svm_bow:.4f}')
print(f'Precision Recall F1 (SVM, TF-IDF): {precision_svm_tfidf:.4f} {recall_svm_tfidf:.4f} {f1_svm_tfidf:.4f}')
```

```
Precision Recall F1 (SVM, BoW): 0.8591 0.8224 0.8403
Precision Recall F1 (SVM, TF-IDF): 0.8597 0.8577 0.8587
time: 43 s (started: 2023-09-18 06:11:13 +00:00)
```

With Pos Tag Lem

```
# Train and evaluate SVM model using BoW features
precision_svm_bow, recall_svm_bow, f1_svm_bow = train_and_evaluate_model(
    LinearSVC,
    X_train_bow_pt, y_train, X_test_bow_pt, y_test,
    max_iter=2000
)

# Train and evaluate SVM model using TF-IDF features
precision_svm_tfidf, recall_svm_tfidf, f1_svm_tfidf = train_and_evaluate_model(
    LinearSVC,
    X_train_tfidf_pt, y_train, X_test_tfidf_pt, y_test,
    max_iter=2000
)

# Print the results
print(f'Precision Recall F1 (SVM, BoW): {precision_svm_bow:.4f} {recall_svm_bow:.4f} {f1_svm_bow:.4f}')
print(f'Precision Recall F1 (SVM, TF-IDF): {precision_svm_tfidf:.4f} {recall_svm_tfidf:.4f} {f1_svm_tfidf:.4f}')
```

```
Precision Recall F1 (SVM, BoW): 0.8538 0.8186 0.8358
Precision Recall F1 (SVM, TF-IDF): 0.8564 0.8544 0.8554
time: 47.1 s (started: 2023-09-18 06:03:37 +00:00)
```

Logistic Regression Using Both Features

Without Pos Tag Lem

```
# Train and evaluate Logistic Regression model using BoW features
precision_lr_bow, recall_lr_bow, f1_lr_bow = train_and_evaluate_model(
    LogisticRegression,
    X_train_bow, y_train, X_test_bow, y_test,
    max_iter=2000
)

# Train and evaluate Logistic Regression model using TF-IDF features
precision_lr_tfidf, recall_lr_tfidf, f1_lr_tfidf = train_and_evaluate_model(
    LogisticRegression,
    X_train_tfidf, y_train, X_test_tfidf, y_test,
    max_iter=2000
)

# Print the results
print(f'Precision Recall F1 (Logistic Regression, BoW): {precision_lr_bow:.4f} {recall_lr_bow:.4f} {f1_lr_bow:.4f}')
print(f'Precision Recall F1 (Logistic Regression, TF-IDF): {precision_lr_tfidf:.4f} {recall_lr_tfidf:.4f} {f1_lr_tfidf:.4f}')
```

```
Precision Recall F1 (Logistic Regression, BoW): 0.8694 0.8358 0.8523
Precision Recall F1 (Logistic Regression, TF-IDF): 0.8582 0.8641 0.8611
time: 8.34 s (started: 2023-09-18 06:15:55 +00:00)
```

With Pos Tag Lem

```
# Train and evaluate Logistic Regression model using BoW features
precision_lr_bow, recall_lr_bow, f1_lr_bow = train_and_evaluate_model(
    LogisticRegression,
    X_train_bow_pt, y_train, X_test_bow_pt, y_test,
    max_iter=2000
)

# Train and evaluate Logistic Regression model using TF-IDF features
precision_lr_tfidf, recall_lr_tfidf, f1_lr_tfidf = train_and_evaluate_model(
    LogisticRegression,
    X_train_tfidf_pt, y_train, X_test_tfidf_pt, y_test,
    max_iter=2000
```

```
)

# Print the results
print(f'Precision Recall F1 (Logistic Regression, BoW): {precision_lr_bow:.4f} {recall_lr_bow:.4f} {f1_lr_bow:.4f}')
print(f'Precision Recall F1 (Logistic Regression, TF-IDF): {precision_lr_tfidf:.4f} {recall_lr_tfidf:.4f} {f1_lr_tfidf:.4f}')
```

```
Precision Recall F1 (Logistic Regression, BoW): 0.8657 0.8307 0.8478
Precision Recall F1 (Logistic Regression, TF-IDF): 0.8562 0.8606 0.8584
time: 7.83 s (started: 2023-09-18 06:16:09 +00:00)
```

Naive Bayes Using Both Features

Without Pos Tag Lem

```
# Train and evaluate Naive Bayes model using BoW features
precision_nb_bow, recall_nb_bow, f1_nb_bow = train_and_evaluate_model(
    MultinomialNB,
    X_train_bow, y_train, X_test_bow, y_test
)

# Train and evaluate Naive Bayes model using TF-IDF features
precision_nb_tfidf, recall_nb_tfidf, f1_nb_tfidf = train_and_evaluate_model(
    MultinomialNB,
    X_train_tfidf, y_train, X_test_tfidf, y_test
)

# Print the results
print(f'Precision Recall F1 (Naive Bayes, BoW): {precision_nb_bow:.4f} {recall_nb_bow:.4f} {f1_nb_bow:.4f}')
print(f'Precision Recall F1 (Naive Bayes, TF-IDF): {precision_nb_tfidf:.4f} {recall_nb_tfidf:.4f} {f1_nb_tfidf:.4f}')
```

```
Precision Recall F1 (Naive Bayes, BoW): 0.8496 0.7718 0.8089
Precision Recall F1 (Naive Bayes, TF-IDF): 0.8284 0.8279 0.8281
time: 224 ms (started: 2023-09-18 06:17:30 +00:00)
```

With Pos Tag Lem

```
# Train and evaluate Naive Bayes model using BoW features
precision_nb_bow, recall_nb_bow, f1_nb_bow = train_and_evaluate_model(
    MultinomialNB,
    X_train_bow_pt, y_train, X_test_bow_pt, y_test
)

# Train and evaluate Naive Bayes model using TF-IDF features
precision_nb_tfidf, recall_nb_tfidf, f1_nb_tfidf = train_and_evaluate_model(
    MultinomialNB,
    X_train_tfidf_pt, y_train, X_test_tfidf_pt, y_test
)

# Print the results
print(f'Precision Recall F1 (Naive Bayes, BoW): {precision_nb_bow:.4f} {recall_nb_bow:.4f} {f1_nb_bow:.4f}')
print(f'Precision Recall F1 (Naive Bayes, TF-IDF): {precision_nb_tfidf:.4f} {recall_nb_tfidf:.4f} {f1_nb_tfidf:.4f}')
```

```
Precision Recall F1 (Naive Bayes, BoW): 0.8456 0.7678 0.8048
Precision Recall F1 (Naive Bayes, TF-IDF): 0.8234 0.8242 0.8238
time: 127 ms (started: 2023-09-18 06:17:32 +00:00)
```

Convert to Python File

```
!python --version
```

```
Python 3.10.12
time: 108 ms (started: 2023-09-18 06:17:46 +00:00)
```

With negative stopwords

```
%%writefile HW1-CSCI544-w-neg-sw.py
# Python Version: 3.10.12

# Python Version: 3.10.12

import re
import unicodedata

import warnings

warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd

import nltk
from nltk.corpus import stopwords, wordnet
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize

nltk.download("punkt", quiet=True)
nltk.download("wordnet", quiet=True)
nltk.download("stopwords", quiet=True)
nltk.download("averaged_perceptron_tagger", quiet=True)

import contractions

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.metrics import precision_score, recall_score, f1_score

from sklearn.linear_model import Perceptron, LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB


class Config:
    RANDOM_STATE = 56
    DATA_PATH = "amazon_reviews_us_Office_Products_v1_00.tsv.gz"
    TEST_SPLIT = 0.2
    N_SAMPLES_EACH_CLASS = 50000
    NUM_TFIDF_FEATURES = 5000
    NUM_BOW_FEATURES = 5000
```

```python
class DataLoader:
    @staticmethod
    def load_data(path):
        df = pd.read_csv(
            path,
            sep="\t",
            usecols=["review_headline", "review_body", "star_rating"],
            on_bad_lines="skip",
            memory_map=True,
        )
        return df


class DataProcessor:
    @staticmethod
    def filter_columns(df):
        return df.loc[:, ["review_body", "star_rating"]]

    @staticmethod
    def convert_star_rating(df):
        df["star_rating"] = pd.to_numeric(df["star_rating"], errors="coerce")
        df.dropna(subset=["star_rating"], inplace=True)
        return df

    @staticmethod
    def classify_sentiment(df):
        df["sentiment"] = df["star_rating"].apply(lambda x: 1 if x <= 3 else 2)
        return df

    @staticmethod
    def sample_data(df, n_samples, random_state):
        sampled_df = pd.concat(
            [
                df.query("sentiment==1").sample(n=n_samples, random_state=random_state),
                df.query("sentiment==2").sample(n=n_samples, random_state=random_state),
            ],
            ignore_index=True,
        ).sample(frac=1, random_state=random_state)

        sampled_df.drop(columns=["star_rating"], inplace=True)
        return sampled_df


class TextCleaner:
    @staticmethod
    def unicode_to_ascii(s):
        return "".join(
            c for c in unicodedata.normalize("NFD", s) if unicodedata.category(c) != "Mn"
        )

    @staticmethod
    def expand_contractions(text):
        return contractions.fix(text)

    @staticmethod
    def remove_email_addresses(text):
        return re.sub(r"[a-zA-Z0-9_\-\.]+@[a-zA-Z0-9_\-\.]+\.[a-zA-Z]{2,5}", " ", text)

    @staticmethod
    def remove_urls(text):
        return re.sub(r"\bhttps?:\/\/\S+|www\.\S+", " ", text)

    @staticmethod
    def remove_html_tags(text):
        return re.sub(r"<.*?>", "", text)

    @staticmethod
    def clean_text(text):
        text = TextCleaner.unicode_to_ascii(text.lower().strip())
        # replacing email addresses with empty string
        text = TextCleaner.remove_email_addresses(text)
        # replacing urls with empty string
        text = TextCleaner.remove_urls(text)
        # Remove HTML tags
        text = TextCleaner.remove_html_tags(text)
        # Expand contraction for eg., wouldn't => would not
        text = TextCleaner.expand_contractions(text)
        # creating a space between a word and the punctuation following it
        text = re.sub(r"([?.!,¿])", r" \1 ", text)
        text = re.sub(r'[" "]+', " ", text)
        # removes all non-alphabetical characters
        text = re.sub(r"[^a-zA-Z\s]+", "", text)
        # remove extra spaces
        text = re.sub(" +", " ", text)
        text = text.strip()
        return text


class TextPreprocessor:
    lemmatizer = WordNetLemmatizer()

    @staticmethod
    def get_stopwords_pattern():
        # Stopword list
        og_stopwords = set(stopwords.words("english"))

        # Define a list of negative words to remove
        # neg_words = ["no", "not", "nor", "neither", "none", "never", "nobody", "nowhere"]
        # custom_stopwords = [word for word in og_stopwords if word not in neg_words]
        pattern = re.compile(r"\b(" + r"|".join(og_stopwords) + r")\b\s*")
        return pattern

    @staticmethod
    def pos_tagger(tag):
        if tag.startswith("J"):
            return wordnet.ADJ
        elif tag.startswith("V"):
            return wordnet.VERB
        elif tag.startswith("N"):
            return wordnet.NOUN
        elif tag.startswith("R"):
            return wordnet.ADV
        else:
            return None

    @staticmethod
    def lemmatize_text_using_pos_tags(text):
```

```
        words = nltk.pos_tag(word_tokenize(text))
        words = map(lambda x: (x[0], TextPreprocessor.pos_tagger(x[1])), words)
        lemmatized_words = [
            TextPreprocessor.lemmatizer.lemmatize(word, tag) if tag else word for word, tag in words
        ]
        return " ".join(lemmatized_words)

    @staticmethod
    def lemmatize_text(text):
        words = word_tokenize(text)
        lemmatized_words = [TextPreprocessor.lemmatizer.lemmatize(word) for word in words]
        return " ".join(lemmatized_words)

    pattern = get_stopwords_pattern()

    @staticmethod
    def preprocess_text(text):
        # replacing all the stopwords
        text = TextPreprocessor.pattern.sub("", text)
        text = TextPreprocessor.lemmatize_text(text)
        return text


clean_text_vect = np.vectorize(TextCleaner.clean_text)
preprocess_text_vect = np.vectorize(TextPreprocessor.preprocess_text)


def clean_and_process_data(path):
    df = DataLoader.load_data(path)
    df_filtered = DataProcessor.filter_columns(df)
    df_filtered = DataProcessor.convert_star_rating(df_filtered)
    df_filtered = DataProcessor.classify_sentiment(df_filtered)

    balanced_df = DataProcessor.sample_data(
        df_filtered, Config.N_SAMPLES_EACH_CLASS, Config.RANDOM_STATE
    )

    balanced_df["review_body"] = balanced_df["review_body"].astype(str)

    # Clean data
    avg_len_before_clean = balanced_df["review_body"].apply(len).mean()
    balanced_df["review_body"] = balanced_df["review_body"].apply(clean_text_vect)
    # Drop reviews that are empty
    balanced_df = balanced_df.loc[balanced_df["review_body"].str.strip() != ""]
    avg_len_after_clean = balanced_df["review_body"].apply(len).mean()

    # Preprocess data
    avg_len_before_preprocess = avg_len_after_clean
    balanced_df["review_body"] = balanced_df["review_body"].apply(preprocess_text_vect)
    avg_len_after_preprocess = balanced_df["review_body"].apply(len).mean()

    # Print Results
    print(f"{avg_len_before_clean:.2f}, {avg_len_after_clean:.2f}")
    print(f"{avg_len_before_preprocess:.2f}, {avg_len_after_preprocess:.2f}")

    return balanced_df


def evaluate_model(model, X_test, y_test):
    # Predict on the test set
    y_pred = model.predict(X_test)

    # Calculate evaluation metrics
    precision = precision_score(y_test, y_pred, average="binary")
    recall = recall_score(y_test, y_pred, average="binary")
    f1 = f1_score(y_test, y_pred, average="binary")

    return precision, recall, f1


def train_and_evaluate_model(model_class, X_train, y_train, X_test, y_test, **model_params):
    # Initialize model
    model = model_class(**model_params)

    # Train the model
    model.fit(X_train, y_train)

    # Evaluate model
    precision, recall, f1 = evaluate_model(model, X_test, y_test)
    return model, precision, recall, f1


def main():
    balanced_df = clean_and_process_data(Config.DATA_PATH)

    # Splitting the reviews dataset
    X_train, X_test, y_train, y_test = train_test_split(
        balanced_df["review_body"],
        balanced_df["sentiment"],
        test_size=Config.TEST_SPLIT,
        random_state=Config.RANDOM_STATE,
    )

    # Feature Extraction
    tfidf_vectorizer = TfidfVectorizer(max_features=Config.NUM_TFIDF_FEATURES)
    X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
    X_test_tfidf = tfidf_vectorizer.transform(X_test)

    count_vectorizer = CountVectorizer(max_features=Config.NUM_BOW_FEATURES)
    X_train_bow = count_vectorizer.fit_transform(X_train)
    X_test_bow = count_vectorizer.transform(X_test)

    # Train and evaluate Perceptron model using BoW features
    (
        _,
        precision_perceptron_bow,
        recall_perceptron_bow,
        f1_perceptron_bow,
    ) = train_and_evaluate_model(
        Perceptron, X_train_bow, y_train, X_test_bow, y_test, max_iter=4000
    )

    # Train and evaluate Perceptron model using TF-IDF features
    (
        _,
        precision_perceptron_tfidf,
        recall_perceptron_tfidf,
        f1_perceptron_tfidf,
    ) = train_and_evaluate_model(
```

```
        Perceptron, X_train_tfidf, y_train, X_test_tfidf, y_test, max_iter=4000
    )

    # Train and evaluate SVM model using BoW features
    _, precision_svm_bow, recall_svm_bow, f1_svm_bow = train_and_evaluate_model(
        LinearSVC, X_train_bow, y_train, X_test_bow, y_test, max_iter=2500
    )

    # Train and evaluate SVM model using TF-IDF features
    _, precision_svm_tfidf, recall_svm_tfidf, f1_svm_tfidf = train_and_evaluate_model(
        LinearSVC, X_train_tfidf, y_train, X_test_tfidf, y_test, max_iter=2500
    )

    # Train and evaluate Logistic Regression model using BoW features
    _, precision_lr_bow, recall_lr_bow, f1_lr_bow = train_and_evaluate_model(
        LogisticRegression, X_train_bow, y_train, X_test_bow, y_test, max_iter=4000
    )

    # Train and evaluate Logistic Regression model using TF-IDF features
    _, precision_lr_tfidf, recall_lr_tfidf, f1_lr_tfidf = train_and_evaluate_model(
        LogisticRegression, X_train_tfidf, y_train, X_test_tfidf, y_test, max_iter=4000
    )

    # Train and evaluate Naive Bayes model using BoW features
    _, precision_nb_bow, recall_nb_bow, f1_nb_bow = train_and_evaluate_model(
        MultinomialNB, X_train_bow, y_train, X_test_bow, y_test
    )

    # Train and evaluate Naive Bayes model using TF-IDF features
    _, precision_nb_tfidf, recall_nb_tfidf, f1_nb_tfidf = train_and_evaluate_model(
        MultinomialNB, X_train_tfidf, y_train, X_test_tfidf, y_test
    )

    # Print the results
    print(f"{precision_perceptron_bow:.4f} {recall_perceptron_bow:.4f} {f1_perceptron_bow:.4f}")
    print(
        f"{precision_perceptron_tfidf:.4f} {recall_perceptron_tfidf:.4f} {f1_perceptron_tfidf:.4f}"
    )

    print(f"{precision_svm_bow:.4f} {recall_svm_bow:.4f} {f1_svm_bow:.4f}")
    print(f"{precision_svm_tfidf:.4f} {recall_svm_tfidf:.4f} {f1_svm_tfidf:.4f}")

    print(f"{precision_lr_bow:.4f} {recall_lr_bow:.4f} {f1_lr_bow:.4f}")
    print(f"{precision_lr_tfidf:.4f} {recall_lr_tfidf:.4f} {f1_lr_tfidf:.4f}")

    print(f"{precision_nb_bow:.4f} {recall_nb_bow:.4f} {f1_nb_bow:.4f}")
    print(f"{precision_nb_tfidf:.4f} {recall_nb_tfidf:.4f} {f1_nb_tfidf:.4f}")


if __name__ == "__main__":
    main()
```

```
Writing HW1-CSCI544-w-neg-sw.py
time: 15.9 ms (started: 2023-09-18 06:50:31 +00:00)
```

Without negative stopwords

```
%%writefile HW1-CSCI544-wo-neg-sw.py
# Python Version: 3.10.12

import re
import unicodedata

import warnings

warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd

import nltk
from nltk.corpus import stopwords, wordnet
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize

nltk.download("punkt", quiet=True)
nltk.download("wordnet", quiet=True)
nltk.download("stopwords", quiet=True)
nltk.download("averaged_perceptron_tagger", quiet=True)

import contractions

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.metrics import precision_score, recall_score, f1_score

from sklearn.linear_model import Perceptron, LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB


class Config:
    RANDOM_STATE = 56
    DATA_PATH = "amazon_reviews_us_Office_Products_v1_00.tsv.gz"
    TEST_SPLIT = 0.2
    N_SAMPLES_EACH_CLASS = 50000
    NUM_TFIDF_FEATURES = 5000
    NUM_BOW_FEATURES = 5000


class DataLoader:
    @staticmethod
    def load_data(path):
        df = pd.read_csv(
            path,
            sep="\t",
            usecols=["review_headline", "review_body", "star_rating"],
            on_bad_lines="skip",
            memory_map=True,
        )
        return df


class DataProcessor:
    @staticmethod
    def filter_columns(df):
        return df.loc[:, ["review_body", "star_rating"]]
```

```python

    @staticmethod
    def convert_star_rating(df):
        df["star_rating"] = pd.to_numeric(df["star_rating"], errors="coerce")
        df.dropna(subset=["star_rating"], inplace=True)
        return df

    @staticmethod
    def classify_sentiment(df):
        df["sentiment"] = df["star_rating"].apply(lambda x: 1 if x <= 3 else 2)
        return df

    @staticmethod
    def sample_data(df, n_samples, random_state):
        sampled_df = pd.concat(
            [
                df.query("sentiment==1").sample(n=n_samples, random_state=random_state),
                df.query("sentiment==2").sample(n=n_samples, random_state=random_state),
            ],
            ignore_index=True,
        ).sample(frac=1, random_state=random_state)

        sampled_df.drop(columns=["star_rating"], inplace=True)
        return sampled_df


class TextCleaner:
    @staticmethod
    def unicode_to_ascii(s):
        return "".join(
            c for c in unicodedata.normalize("NFD", s) if unicodedata.category(c) != "Mn"
        )

    @staticmethod
    def expand_contractions(text):
        return contractions.fix(text)

    @staticmethod
    def remove_email_addresses(text):
        return re.sub(r"[a-zA-Z0-9_\-\.]+@[a-zA-Z0-9_\-\.]+\.[a-zA-Z]{2,5}", " ", text)

    @staticmethod
    def remove_urls(text):
        return re.sub(r"\bhttps?:\/\/\S+|www\.\S+", " ", text)

    @staticmethod
    def remove_html_tags(text):
        return re.sub(r"<.*?>", "", text)

    @staticmethod
    def clean_text(text):
        text = TextCleaner.unicode_to_ascii(text.lower().strip())
        # replacing email addresses with empty string
        text = TextCleaner.remove_email_addresses(text)
        # replacing urls with empty string
        text = TextCleaner.remove_urls(text)
        # Remove HTML tags
        text = TextCleaner.remove_html_tags(text)
        # Expand contraction for eg., wouldn't => would not
        text = TextCleaner.expand_contractions(text)
        # creating a space between a word and the punctuation following it
        text = re.sub(r"([?.!,¿])", r" \1 ", text)
        text = re.sub(r'[" "]+', " ", text)
        # removes all non-alphabetical characters
        text = re.sub(r"[^a-zA-Z\s]+", "", text)
        # remove extra spaces
        text = re.sub(" +", " ", text)
        text = text.strip()
        return text


class TextPreprocessor:
    lemmatizer = WordNetLemmatizer()

    @staticmethod
    def get_stopwords_pattern():
        # Stopword list
        og_stopwords = set(stopwords.words("english"))

        # Define a list of negative words to remove
        neg_words = ["no", "not", "nor", "neither", "none", "never", "nobody", "nowhere"]
        custom_stopwords = [word for word in og_stopwords if word not in neg_words]
        pattern = re.compile(r"\b(" + r"|".join(custom_stopwords) + r")\b\s*")
        return pattern

    @staticmethod
    def pos_tagger(tag):
        if tag.startswith("J"):
            return wordnet.ADJ
        elif tag.startswith("V"):
            return wordnet.VERB
        elif tag.startswith("N"):
            return wordnet.NOUN
        elif tag.startswith("R"):
            return wordnet.ADV
        else:
            return None

    @staticmethod
    def lemmatize_text_using_pos_tags(text):
        words = nltk.pos_tag(word_tokenize(text))
        words = map(lambda x: (x[0], TextPreprocessor.pos_tagger(x[1])), words)
        lemmatized_words = [
            TextPreprocessor.lemmatizer.lemmatize(word, tag) if tag else word for word, tag in words
        ]
        return " ".join(lemmatized_words)

    @staticmethod
    def lemmatize_text(text):
        words = word_tokenize(text)
        lemmatized_words = [TextPreprocessor.lemmatizer.lemmatize(word) for word in words]
        return " ".join(lemmatized_words)

    pattern = get_stopwords_pattern()

    @staticmethod
    def preprocess_text(text):
        # replacing all the stopwords
        text = TextPreprocessor.pattern.sub("", text)
```

```python
        text = TextPreprocessor.lemmatize_text(text)
        return text


clean_text_vect = np.vectorize(TextCleaner.clean_text)
preprocess_text_vect = np.vectorize(TextPreprocessor.preprocess_text)


def clean_and_process_data(path):
    df = DataLoader.load_data(path)
    df_filtered = DataProcessor.filter_columns(df)
    df_filtered = DataProcessor.convert_star_rating(df_filtered)
    df_filtered = DataProcessor.classify_sentiment(df_filtered)

    balanced_df = DataProcessor.sample_data(
        df_filtered, Config.N_SAMPLES_EACH_CLASS, Config.RANDOM_STATE
    )

    balanced_df["review_body"] = balanced_df["review_body"].astype(str)

    # Clean data
    avg_len_before_clean = balanced_df["review_body"].apply(len).mean()
    balanced_df["review_body"] = balanced_df["review_body"].apply(clean_text_vect)
    # Drop reviews that are empty
    balanced_df = balanced_df.loc[balanced_df["review_body"].str.strip() != ""]
    avg_len_after_clean = balanced_df["review_body"].apply(len).mean()

    # Preprocess data
    avg_len_before_preprocess = avg_len_after_clean
    balanced_df["review_body"] = balanced_df["review_body"].apply(preprocess_text_vect)
    avg_len_after_preprocess = balanced_df["review_body"].apply(len).mean()

    # Print Results
    print(f"{avg_len_before_clean:.2f}, {avg_len_after_clean:.2f}")
    print(f"{avg_len_before_preprocess:.2f}, {avg_len_after_preprocess:.2f}")

    return balanced_df


def evaluate_model(model, X_test, y_test):
    # Predict on the test set
    y_pred = model.predict(X_test)

    # Calculate evaluation metrics
    precision = precision_score(y_test, y_pred, average="binary")
    recall = recall_score(y_test, y_pred, average="binary")
    f1 = f1_score(y_test, y_pred, average="binary")

    return precision, recall, f1


def train_and_evaluate_model(model_class, X_train, y_train, X_test, y_test, **model_params):
    # Initialize model
    model = model_class(**model_params)

    # Train the model
    model.fit(X_train, y_train)

    # Evaluate model
    precision, recall, f1 = evaluate_model(model, X_test, y_test)
    return model, precision, recall, f1


def main():
    balanced_df = clean_and_process_data(Config.DATA_PATH)

    # Splitting the reviews dataset
    X_train, X_test, y_train, y_test = train_test_split(
        balanced_df["review_body"],
        balanced_df["sentiment"],
        test_size=Config.TEST_SPLIT,
        random_state=Config.RANDOM_STATE,
    )

    # Feature Extraction
    tfidf_vectorizer = TfidfVectorizer(max_features=Config.NUM_TFIDF_FEATURES)
    X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
    X_test_tfidf = tfidf_vectorizer.transform(X_test)

    count_vectorizer = CountVectorizer(max_features=Config.NUM_BOW_FEATURES)
    X_train_bow = count_vectorizer.fit_transform(X_train)
    X_test_bow = count_vectorizer.transform(X_test)

    # Train and evaluate Perceptron model using BoW features
    (
        _,
        precision_perceptron_bow,
        recall_perceptron_bow,
        f1_perceptron_bow,
    ) = train_and_evaluate_model(
        Perceptron, X_train_bow, y_train, X_test_bow, y_test, max_iter=4000
    )

    # Train and evaluate Perceptron model using TF-IDF features
    (
        _,
        precision_perceptron_tfidf,
        recall_perceptron_tfidf,
        f1_perceptron_tfidf,
    ) = train_and_evaluate_model(
        Perceptron, X_train_tfidf, y_train, X_test_tfidf, y_test, max_iter=4000
    )

    # Train and evaluate SVM model using BoW features
    _, precision_svm_bow, recall_svm_bow, f1_svm_bow = train_and_evaluate_model(
        LinearSVC, X_train_bow, y_train, X_test_bow, y_test, max_iter=2500
    )

    # Train and evaluate SVM model using TF-IDF features
    _, precision_svm_tfidf, recall_svm_tfidf, f1_svm_tfidf = train_and_evaluate_model(
        LinearSVC, X_train_tfidf, y_train, X_test_tfidf, y_test, max_iter=2500
    )

    # Train and evaluate Logistic Regression model using BoW features
    _, precision_lr_bow, recall_lr_bow, f1_lr_bow = train_and_evaluate_model(
        LogisticRegression, X_train_bow, y_train, X_test_bow, y_test, max_iter=4000
    )

    # Train and evaluate Logistic Regression model using TF-IDF features
    _, precision_lr_tfidf, recall_lr_tfidf, f1_lr_tfidf = train_and_evaluate_model(
```

```
    _, precision_lr_tfidf, recall_lr_tfidf, f1_lr_tfidf = train_and_evaluate_model(
        LogisticRegression, X_train_tfidf, y_train, X_test_tfidf, y_test, max_iter=4000
    )

    # Train and evaluate Naive Bayes model using BoW features
    _, precision_nb_bow, recall_nb_bow, f1_nb_bow = train_and_evaluate_model(
        MultinomialNB, X_train_bow, y_train, X_test_bow, y_test
    )

    # Train and evaluate Naive Bayes model using TF-IDF features
    _, precision_nb_tfidf, recall_nb_tfidf, f1_nb_tfidf = train_and_evaluate_model(
        MultinomialNB, X_train_tfidf, y_train, X_test_tfidf, y_test
    )

    # Print the results
    print(f"{precision_perceptron_bow:.4f} {recall_perceptron_bow:.4f} {f1_perceptron_bow:.4f}")
    print(
        f"{precision_perceptron_tfidf:.4f} {recall_perceptron_tfidf:.4f} {f1_perceptron_tfidf:.4f}"
    )

    print(f"{precision_svm_bow:.4f} {recall_svm_bow:.4f} {f1_svm_bow:.4f}")
    print(f"{precision_svm_tfidf:.4f} {recall_svm_tfidf:.4f} {f1_svm_tfidf:.4f}")

    print(f"{precision_lr_bow:.4f} {recall_lr_bow:.4f} {f1_lr_bow:.4f}")
    print(f"{precision_lr_tfidf:.4f} {recall_lr_tfidf:.4f} {f1_lr_tfidf:.4f}")

    print(f"{precision_nb_bow:.4f} {recall_nb_bow:.4f} {f1_nb_bow:.4f}")
    print(f"{precision_nb_tfidf:.4f} {recall_nb_tfidf:.4f} {f1_nb_tfidf:.4f}")


if __name__ == "__main__":
    main()
```

```
Writing HW1-CSCI544-wo-neg-sw.py
time: 9.04 ms (started: 2023-09-18 06:50:34 +00:00)
```

Comparing Results

```
!python HW1-CSCI544-w-neg-sw.py
```

```
314.91, 299.82
299.82, 185.41
0.8007 0.7659 0.7829
0.7655 0.8360 0.7992
0.8521 0.8111 0.8311
0.8454 0.8442 0.8448
0.8525 0.8214 0.8367
0.8461 0.8572 0.8516
0.8467 0.7488 0.7947
0.8291 0.8189 0.8240
time: 4min 11s (started: 2023-09-18 06:50:52 +00:00)
```

```
!python HW1-CSCI544-wo-neg-sw.py
```

```
314.91, 299.82
299.82, 189.65
0.8368 0.7812 0.8080
0.7637 0.8702 0.8135
0.8649 0.8284 0.8463
0.8573 0.8602 0.8588
0.8680 0.8343 0.8508
0.8574 0.8674 0.8624
0.8527 0.7718 0.8102
0.8334 0.8291 0.8312
time: 4min 33s (started: 2023-09-18 06:58:06 +00:00)
```

Conclusions

- Performance models improved after removing negative stopwords from stopwords list
- Performance of models was almost same when trained with dataset with and without pos tagging
- Exploration using Grid search found list to parameters same as the default ones with best estimator performing the same as the default model

THE END