Dependencies

Install

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

```
Requirement already satisfied: contractions in /usr/local/lib/python3.10/dist-packages (0.1.73)
Requirement already satisfied: textsearch>=0.0.21 in /usr/local/lib/python3.10/dist-packages (from contractions) (0.0.24)
Requirement already satisfied: anyascii in /usr/local/lib/python3.10/dist-packages (from textsearch>=0.0.21->contractions) (0.3.2)
Requirement already satisfied: pyahocorasick in /usr/local/lib/python3.10/dist-packages (from textsearch>=0.0.21->contractions) (2.0.0)
Requirement already satisfied: ipython-autotime in /usr/local/lib/python3.10/dist-packages (0.3.1)
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)
Requirement already satisfied: jedi>=0.16 in /usr/local/lib/python3.10/dist-packages (from ipython->ipython-autotime) (0.19.0)
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
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-autot
```

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
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize

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

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
```

```
time: 477 µs (started: 2023-09-13 00:07:49 +00:00)
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

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 = 5000
```

```
time: 678 µs (started: 2023-09-13 00:07:49 +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: 364 µs (started: 2023-09-13 00:07:49 +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: 3.23 ms (started: 2023-09-13 00:07:49 +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: 166 ms (started: 2023-09-13 00:07:49 +00:00)
```

```
del df_small
```

```
time: 724 µs (started: 2023-09-13 00:07:49 +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: 48.2 s (started: 2023-09-13 00:07:49 +00:00)
```

```
df.head()
```

	star_rating	review_headline	review_body
0	5	Five Stars	Great product.
1	5	Phffffffft, Phffffffft. 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: 106 ms (started: 2023-09-13 00:08:38 +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.

```python
# 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: 103 ms (started: 2023-09-13 00:08:38 +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: 3.65 s (started: 2023-09-13 00:08:38 +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

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 abbr like "I'll" to "I will"

We vectorize the clean_text function for better performance

```
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: 4.91 ms (started: 2023-09-13 00:08:43 +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)

# 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: 299.72 characters
time: 52.3 s (started: 2023-09-13 00:08:43 +00:00)
```

Pre-processing

- Remove the stopwords
  - Do not exclude negative stopwords
- Lemmatize words after tokenization

Vectorize the preprocess_text function for better performance

```
# Stopword list
og_stopwords = set(stopwords.words('english'))
```

```python
# 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*')

def lemmatize_text(text):
    lemmatizer = WordNetLemmatizer()
    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

preprocess_text_vect = np.vectorize(preprocess_text)
```

```
time: 26 ms (started: 2023-09-13 00:10:17 +00:00)
```

```python
# Calculate average length of reviews before cleaning (combined)
avg_len_before_preprocess = avg_len_after_clean

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

# Calculate average length of reviews after cleaning (combined)
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: 299.72 characters
Avg. Length of Reviews After Preprocessing: 189.58 characters
time: 1min 54s (started: 2023-09-13 00:10:21 +00:00)
```

Train and Test Split

```python
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
)
```

```
time: 34.1 ms (started: 2023-09-13 00:12:20 +00:00)
```

Feature Extraction

TF-IDF

```python
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)
```

```
time: 3.07 s (started: 2023-09-13 00:12:22 +00:00)
```

Bag of words

```python
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)
```

```
time: 4.21 s (started: 2023-09-13 00:12:25 +00:00)
```

ML Modeling

```python
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
```

```
time: 828 µs (started: 2023-09-13 00:12:29 +00:00)
```

Perceptron Using Both Features

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],
    'tol': [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=3  # Number of cross-validation folds
)

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}')
```

```
Best Parameters (BoW): {'max_iter': 1000, 'penalty': 'l1', 'tol': 0.001}
Precision Recall F1 (Perceptron, BoW): 0.8295 0.8024 0.8157
time: 1min 4s (started: 2023-09-13 00:12:55 +00:00)
```

```
# Grid search for TF-IDF features
grid_search_tfidf = GridSearchCV(
    estimator=perceptron,
    param_grid=param_grid,
    scoring='f1',
    cv=3  # Number of cross-validation folds
)

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}')
```

```
Best Parameters (TF-IDF): {'max_iter': 1000, 'penalty': 'elasticnet', 'tol': 0.001}
Precision Recall F1 (Perceptron, TF-IDF): 0.8250 0.7294 0.7742
time: 26.2 s (started: 2023-09-13 00:14:00 +00:00)
```

```
def train_evaluate_perceptron(X_train, y_train, X_test, y_test):
    # Initialize Perceptron model
    perceptron = Perceptron(max_iter=4000)

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

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


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

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

# 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): 0.8354 0.7942 0.8143
Precision Recall F1 (TF-IDF): 0.7907 0.8249 0.8075
time: 651 ms (started: 2023-09-13 00:14:26 +00:00)
```

SVM Using Both Features

```
def train_evaluate_svm(X_train, y_train, X_test, y_test):
    # Initialize SVM model
    svm = LinearSVC(max_iter=2000)

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

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

# Train and evaluate SVM model using BoW features
precision_svm_bow, recall_svm_bow, f1_svm_bow = train_evaluate_svm(X_train_bow, y_train, X_test_bow, y_test)

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

# 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): 0.8684 0.8340 0.8509
Precision Recall F1 (SVM, TF-IDF): 0.8585 0.8604 0.8594
time: 45.9 s (started: 2023-09-13 00:14:27 +00:00)
```

Logistic Regression Using Both Features

```
def train_evaluate_logistic_regression(X_train, y_train, X_test, y_test):
    # Initialize Logistic Regression model
    log_reg = LogisticRegression(max_iter=2000)

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

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

    return precision, recall, f1

# Train and evaluate Logistic Regression model using BoW features
precision_lr_bow, recall_lr_bow, f1_lr_bow = train_evaluate_logistic_regression(X_train_bow, y_train, X_test_bow, y_test)

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

# 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): 0.8710 0.8419 0.8562
Precision Recall F1 (Logistic Regression, TF-IDF): 0.8592 0.8680 0.8636
time: 7.1 s (started: 2023-09-13 00:15:13 +00:00)
```

Naive Bayes Using Both Features

```
def train_evaluate_naive_bayes(X_train, y_train, X_test, y_test):
    # Initialize Naive Bayes model (Multinomial Naive Bayes for text classification)
    nb_model = MultinomialNB()

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

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

    return precision, recall, f1

# Train and evaluate Naive Bayes model using BoW features
precision_nb_bow, recall_nb_bow, f1_nb_bow = train_evaluate_naive_bayes(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_evaluate_naive_bayes(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): 0.8486 0.7755 0.8104
Precision Recall F1 (Naive Bayes, TF-IDF): 0.8235 0.8302 0.8268
time: 129 ms (started: 2023-09-13 00:15:20 +00:00)
```

Convert to Python File

```
!python --version
```

```
Python 3.10.12
time: 109 ms (started: 2023-09-13 00:18:04 +00:00)
```

```
%%writefile HW1-CSCI544.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
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)

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 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:
    @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
```

```
        return pattern

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

    @staticmethod
    def preprocess_text(text):
        # replacing all the stopwords
        text = TextPreprocessor.get_stopwords_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)


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


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)
    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_evaluate_perceptron(X_train, y_train, X_test, y_test):
    # Initialize Perceptron model
    perceptron = Perceptron(max_iter=4000)

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

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


def train_evaluate_svm(X_train, y_train, X_test, y_test):
    # Initialize SVM model
    svm = LinearSVC(max_iter=2500)

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

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


def train_evaluate_logistic_regression(X_train, y_train, X_test, y_test):
    # Initialize Logistic Regression model
    log_reg = LogisticRegression(max_iter=4000)

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

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

    return precision, recall, f1


def train_evaluate_naive_bayes(X_train, y_train, X_test, y_test):
    # Initialize Naive Bayes model (Multinomial Naive Bayes for text classification)
    nb_model = MultinomialNB()

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

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

    return 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_evaluate_perceptron(
        X_train_bow, y_train, X_test_bow, y_test
    )

    # Train and evaluate Perceptron model using TF-IDF features
    (
        precision_perceptron_tfidf,
        recall_perceptron_tfidf,
        f1_perceptron_tfidf,
    ) = train_evaluate_perceptron(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}"
    )

    # Train and evaluate SVM model using BoW features
    precision_svm_bow, recall_svm_bow, f1_svm_bow = train_evaluate_svm(
```

```
        X_train_bow, y_train, X_test_bow, y_test
    )

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

    # Print the results
    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}")

    # Train and evaluate Logistic Regression model using BoW features
    precision_lr_bow, recall_lr_bow, f1_lr_bow = train_evaluate_logistic_regression(
        X_train_bow, y_train, X_test_bow, y_test
    )

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

    # Print the results
    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}")

    # Train and evaluate Naive Bayes model using BoW features
    precision_nb_bow, recall_nb_bow, f1_nb_bow = train_evaluate_naive_bayes(
        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_evaluate_naive_bayes(
        X_train_tfidf, y_train, X_test_tfidf, y_test
    )

    # Print the results
    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()
```

```
Overwriting HW1-CSCI544.py
time: 15.9 ms (started: 2023-09-13 00:18:13 +00:00)
```

```
!python HW1-CSCI544.py
```

```
314.91, 299.72
299.72, 189.58
0.8354 0.7942 0.8143
0.7907 0.8249 0.8075
0.8684 0.8341 0.8509
0.8585 0.8604 0.8594
0.8710 0.8419 0.8562
0.8592 0.8680 0.8636
0.8486 0.7755 0.8104
0.8235 0.8302 0.8268
time: 5min 9s (started: 2023-09-13 00:18:17 +00:00)
```

THE END

Colab paid products - Cancel contracts here