

▼ Dependencies

▼ Install

```
1 !pip install contractions
2 !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) (3.0.43)
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

```
1 import os
2 import re
3 import unicodedata
4
5 import warnings
6 warnings.filterwarnings("ignore")
7
8 import numpy as np
9 import pandas as pd
10
11 import nltk
12 from nltk.corpus import stopwords
13 from nltk.stem import WordNetLemmatizer
14 from nltk.tokenize import word_tokenize
15
16 nltk.download('punkt')
17 nltk.download('wordnet')
18 nltk.download('stopwords')
19
20 import contractions
21
22 from sklearn.model_selection import train_test_split
23 from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
24 from sklearn.metrics import precision_score, recall_score, f1_score
25
26 from sklearn.linear_model import Perceptron, LogisticRegression
27 from sklearn.svm import SVC, LinearSVC
28 from sklearn.naive_bayes import MultinomialNB
29
30 from sklearn.experimental import enable_halving_search_cv
31 from sklearn.model_selection import HalvingGridSearchCV, GridSearchCV
32
33 %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

```
1 class Config:
2     RANDOM_STATE = 56
3     DATA_PATH = "amazon_reviews_us_Office_Products_v1_00.tsv.gz"
4     TEST_SPLIT = 0.2
5     N_SAMPLES_EACH_CLASS = 50000
6     NUM_TFIDF_FEATURES = 5000
7     NUM_BOW_FEATURES = 5000

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

▼ Download Data

```

1 # %%bash
2 # cd "/content/drive/MyDrive/Colab Notebooks/CSCI544/HW1"
3 # curl -o amazon_reviews_us_Office_Products_v1_00.tsv.gz \
4 # 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.

```
1 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

```

1 df_small = pd.read_csv(
2     Config.DATA_PATH,
3     sep='\t',
4     on_bad_lines="skip",
5     memory_map=True,
6     nrows=10
7 )
8 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)

```

```
1 del df_small
```

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

Read the entire data



```

1 df = pd.read_csv(
2     Config.DATA_PATH,
3     sep='\t',
4     usecols=["review_headline", "review_body", "star_rating"],
5     on_bad_lines="skip",
6     memory_map=True,
7 )

```

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

```
1 df.head()
```



	star_rating	review_headline	review_body	
0	5	Five Stars	Great product.	
1	5	Phfffffft, 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

```

1 # Select columns by name
2 df_filtered = df.loc[:,['review_body', 'star_rating']]
3 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.

```
1 # Check inconsistencies in star_rating column
2 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)
```

```
1 # Convert the 'star_rating' column to numeric, coerce errors to NaN
2 df_filtered['star_rating'] = pd.to_numeric(df['star_rating'], errors='coerce')
3
4 # Drop NaN values from 'star_rating'
5 df_filtered.dropna(subset=["star_rating"], inplace=True)
6
7 # Classify ratings as 1, 2, or 3 into class 1, and ratings 4 and 5 into class 2
8 df_filtered['sentiment'] = df_filtered['star_rating'].apply(
9     lambda x: 1 if x <= 3 else 2
10 )
11
12 print("Shape of unfiltered dataframe:", df.shape)
13 print("Shape of filtered dataframe:", df_filtered.shape)
14
15 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

```
1 # Create a new DataFrame with sampled data
2 balanced_df = pd.concat(
3     [
4         df_filtered.query('sentiment==1').sample(
5             n=Config.N_SAMPLES_EACH_CLASS, random_state=Config.RANDOM_STATE
6         ),
7         df_filtered.query('sentiment==2').sample(
8             n=Config.N_SAMPLES_EACH_CLASS, random_state=Config.RANDOM_STATE
9         )
10    ],
11    ignore_index=True
12 ).sample(frac=1, random_state=Config.RANDOM_STATE)
13
14 balanced_df.drop(columns=["star_rating"], inplace=True)
15
16 # Handling non-string values in Reviews
17 balanced_df["review_body"] = balanced_df["review_body"].astype(str)
18
19 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
 - punctuations
 - 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

```

1 def unicode_to_ascii(s):
2     return ''.join(c for c in unicodedata.normalize('NFD', s)
3         if unicodedata.category(c) != 'Mn')
4
5 def expand_contractions(text):
6     return contractions.fix(text)
7
8 def clean_text(text):
9     text = unicode_to_ascii(text.lower().strip())
10
11     # replacing email addresses with empty string
12     text = re.sub(
13         r"[a-zA-Z0-9_\-\.]+@[a-zA-Z0-9_\-\.]+\.[a-zA-Z]{2,5}", " ", text
14     )
15
16     # replacing urls with empty string
17     text = re.sub(
18         r"\bhttps?:\/\/\S+|www\.\S+", " ", text
19     )
20
21     # Remove HTML tags with empty string
22     text = re.sub(r"<.*?>", "", text)
23
24     # Expand contraction for eg., wouldn't => would not
25     text = expand_contractions(text)
26
27     # creating a space between a word and the punctuation following it
28     text = re.sub(r"([?.!,:;})", r" \1 ", text)
29     text = re.sub(r'" "', " ", text)
30
31     # removes all non-alphabetical characters
32     text = re.sub(r"^[^a-zA-Z\s]+", "", text)
33
34     # remove extra spaces
35     text = re.sub(" +", " ", text)
36
37     text = text.strip()
38     return text
39
40 clean_text_vect = np.vectorize(clean_text)

```

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

```

1 # Calculate average length of reviews before cleaning
2 avg_len_before_clean = balanced_df["review_body"].apply(len).mean()
3
4 balanced_df["review_body"] = balanced_df["review_body"].apply(clean_text_vect)
5
6 # Calculate average length of reviews after cleaning
7 avg_len_after_clean = balanced_df["review_body"].apply(len).mean()
8
9 print(f'Avg. Length of Reviews Before Cleaning: {avg_len_before_clean:.2f} characters')
10 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

```

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

```

```

3 # Define a list of negative words to remove
4 neg_words = ['no', 'not', 'nor', 'neither', 'none', 'never', 'nobody', 'nowhere']
5 custom_stopwords = [word for word in og_stopwords if word not in neg_words]
6
7 pattern = re.compile(r'\b(' + r'|'.join(custom_stopwords) + r')\b\s*')
8
9 def lemmatize_text(text):
10     lemmatizer = WordNetLemmatizer()
11     words = word_tokenize(text)
12     lemmatized_words = [lemmatizer.lemmatize(word) for word in words]
13     return ' '.join(lemmatized_words)
14
15 def preprocess_text(text):
16     # replacing all the stopwords
17     text = pattern.sub('', text)
18     text = lemmatize_text(text)
19     return text
20
21 preprocess_text_vect = np.vectorize(preprocess_text)

```

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

```

1 # Calculate average length of reviews before cleaning (combined)
2 avg_len_before_preprocess = avg_len_after_clean
3
4 balanced_df["review_body"] = balanced_df["review_body"].apply(preprocess_text_vect)
5
6 # Calculate average length of reviews after cleaning (combined)
7 avg_len_after_preprocess = balanced_df["review_body"].apply(len).mean()
8
9 print(f'Avg. Length of Reviews Before Preprocessing: {avg_len_before_preprocess:.2f} characters')
10 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

```

1 X_train, X_test, y_train, y_test = train_test_split(
2     balanced_df['review_body'],
3     balanced_df['sentiment'],
4     test_size=Config.TEST_SPLIT,
5     random_state=Config.RANDOM_STATE
6 )

```

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

▼ Feature Extraction

▼ TF-IDF

```

1 tfidf_vectorizer = TfidfVectorizer(max_features=Config.NUM_TFIDF_FEATURES)
2 X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
3 X_test_tfidf = tfidf_vectorizer.transform(X_test)

```

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

▼ Bag of words

```

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

```

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

▼ ML Modeling

```

1 def evaluate_model(model, X_test, y_test):
2     # Predict on the test set
3     y_pred = model.predict(X_test)
4
5     # Calculate evaluation metrics
6     precision = precision_score(y_test, y_pred, average='binary')
7     recall = recall_score(y_test, y_pred, average='binary')
8     f1 = f1_score(y_test, y_pred, average='binary')
9
10    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

```
1 # Define the parameter grid to search
2 param_grid = {
3     'max_iter': [1000, 2000, 4000, 8000],
4     'tol': [1e-3, 1e-4, 1e-5],
5     'penalty': ['l2', 'l1', 'elasticnet']
6 }
7
8 # Initialize Perceptron model
9 perceptron = Perceptron()
10
11 # Grid search for BoW features
12 grid_search_bow = GridSearchCV(
13     estimator=perceptron,
14     param_grid=param_grid,
15     scoring='f1',
16     cv=3 # Number of cross-validation folds
17 )
18
19 grid_search_bow.fit(X_train_bow, y_train)
20
21 # Get the best parameters and model for BoW
22 best_params_bow = grid_search_bow.best_params_
23 best_model_bow = grid_search_bow.best_estimator_
24
25 # Evaluate the best model for BoW
26 precision_perceptron_bow, recall_perceptron_bow, f1_perceptron_bow = evaluate_model(best_model_bow, X_test_bow, y_test)
27
28 # Print the results for BoW
29 print(f'Best Parameters (BoW): {best_params_bow}')
30 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)
```

```
1 # Grid search for TF-IDF features
2 grid_search_tfidf = GridSearchCV(
3     estimator=perceptron,
4     param_grid=param_grid,
5     scoring='f1',
6     cv=3 # Number of cross-validation folds
7 )
8
9 grid_search_tfidf.fit(X_train_tfidf, y_train)
10
11 # Get the best parameters and model for TF-IDF
12 best_params_tfidf = grid_search_tfidf.best_params_
13 best_model_tfidf = grid_search_tfidf.best_estimator_
14
15 # Evaluate the best model for TF-IDF
16 precision_perceptron_tfidf, recall_perceptron_tfidf, f1_perceptron_tfidf = evaluate_model(best_model_tfidf, X_test_tfidf, y_test)
17
18 # Print the results for TF-IDF
19 print(f'Best Parameters (TF-IDF): {best_params_tfidf}')
20 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)
```

```
1 def train_evaluate_perceptron(X_train, y_train, X_test, y_test):
2     # Initialize Perceptron model
3     perceptron = Perceptron(max_iter=4000)
4
5     # Train the model
6     perceptron.fit(X_train, y_train)
7
8     # Evaluate model
9     precision, recall, f1 = evaluate_model(perceptron, X_test, y_test)
10    return precision, recall, f1
11
12
13 # Train and evaluate Perceptron model using BoW features
14 precision_perceptron_bow, recall_perceptron_bow, f1_perceptron_bow = train_evaluate_perceptron(X_train_bow, y_train, X_test_bow, y_test)
15
16 # Train and evaluate Perceptron model using TF-IDF features
17 precision_perceptron_tfidf, recall_perceptron_tfidf, f1_perceptron_tfidf = train_evaluate_perceptron(X_train_tfidf, y_train, X_test_tfidf, y_test)
18
19 # Print the results
20 print(f'Precision Recall F1 (BoW): {precision_perceptron_bow:.4f} {recall_perceptron_bow:.4f} {f1_perceptron_bow:.4f}')
21 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

```

1 def train_evaluate_svm(X_train, y_train, X_test, y_test):
2     # Initialize SVM model
3     svm = LinearSVC(max_iter=2000)
4
5     # Train the model
6     svm.fit(X_train, y_train)
7
8     # Evaluate model
9     precision, recall, f1 = evaluate_model(svm, X_test, y_test)
10    return precision, recall, f1
11
12 # Train and evaluate SVM model using BoW features
13 precision_svm_bow, recall_svm_bow, f1_svm_bow = train_evaluate_svm(X_train_bow, y_train, X_test_bow, y_test)
14
15 # Train and evaluate SVM model using TF-IDF features
16 precision_svm_tfidf, recall_svm_tfidf, f1_svm_tfidf = train_evaluate_svm(X_train_tfidf, y_train, X_test_tfidf, y_test)
17
18 # Print the results
19 print(f'Precision Recall F1 (SVM, BoW): {precision_svm_bow:.4f} {recall_svm_bow:.4f} {f1_svm_bow:.4f}')
20 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

```

1 def train_evaluate_logistic_regression(X_train, y_train, X_test, y_test):
2     # Initialize Logistic Regression model
3     log_reg = LogisticRegression(max_iter=2000)
4
5     # Train the model
6     log_reg.fit(X_train, y_train)
7
8     # Evaluate model
9     precision, recall, f1 = evaluate_model(log_reg, X_test, y_test)
10
11    return precision, recall, f1
12
13 # Train and evaluate Logistic Regression model using BoW features
14 precision_lr_bow, recall_lr_bow, f1_lr_bow = train_evaluate_logistic_regression(X_train_bow, y_train, X_test_bow, y_test)
15
16 # Train and evaluate Logistic Regression model using TF-IDF features
17 precision_lr_tfidf, recall_lr_tfidf, f1_lr_tfidf = train_evaluate_logistic_regression(X_train_tfidf, y_train, X_test_tfidf, y_test)
18
19 # Print the results
20 print(f'Precision Recall F1 (Logistic Regression, BoW): {precision_lr_bow:.4f} {recall_lr_bow:.4f} {f1_lr_bow:.4f}')
21 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

```

1 def train_evaluate_naive_bayes(X_train, y_train, X_test, y_test):
2     # Initialize Naive Bayes model (Multinomial Naive Bayes for text classification)
3     nb_model = MultinomialNB()
4
5     # Train the model
6     nb_model.fit(X_train, y_train)
7
8     # Evaluate model
9     precision, recall, f1 = evaluate_model(nb_model, X_test, y_test)
10
11    return precision, recall, f1
12
13 # Train and evaluate Naive Bayes model using BoW features
14 precision_nb_bow, recall_nb_bow, f1_nb_bow = train_evaluate_naive_bayes(X_train_bow, y_train, X_test_bow, y_test)
15
16 # Train and evaluate Naive Bayes model using TF-IDF features
17 precision_nb_tfidf, recall_nb_tfidf, f1_nb_tfidf = train_evaluate_naive_bayes(X_train_tfidf, y_train, X_test_tfidf, y_test)
18
19 # Print the results
20 print(f'Precision Recall F1 (Naive Bayes, BoW): {precision_nb_bow:.4f} {recall_nb_bow:.4f} {f1_nb_bow:.4f}')
21 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

```
1 !python --version
```

```

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

```

```

1  %%writefile HW1-CSCI544.py
2  # Python Version: 3.10.12
3
4  import re
5  import unicodedata
6
7  import warnings
8
9  warnings.filterwarnings("ignore")
10
11 import numpy as np
12 import pandas as pd
13
14 import nltk
15 from nltk.corpus import stopwords
16 from nltk.stem import WordNetLemmatizer
17 from nltk.tokenize import word_tokenize
18
19 nltk.download("punkt", quiet=True)
20 nltk.download("wordnet", quiet=True)
21 nltk.download("stopwords", quiet=True)
22
23 import contractions
24
25 from sklearn.model_selection import train_test_split
26 from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
27 from sklearn.metrics import precision_score, recall_score, f1_score
28
29 from sklearn.linear_model import Perceptron, LogisticRegression
30 from sklearn.svm import LinearSVC
31 from sklearn.naive_bayes import MultinomialNB
32
33
34 class Config:
35     RANDOM_STATE = 56
36     DATA_PATH = "amazon_reviews_us_Office_Products_v1_00.tsv.gz"
37     TEST_SPLIT = 0.2
38     N_SAMPLES_EACH_CLASS = 50000
39     NUM_TFIDF_FEATURES = 5000
40     NUM_BOW_FEATURES = 5000
41
42
43 class TextCleaner:
44     @staticmethod
45     def unicode_to_ascii(s):
46         return "".join(
47             c for c in unicodedata.normalize("NFD", s) if unicodedata.category(c) != "Mn"
48         )
49
50     @staticmethod
51     def expand_contractions(text):
52         return contractions.fix(text)
53
54     @staticmethod
55     def remove_email_addresses(text):
56         return re.sub(r"[a-zA-Z0-9_\.]+\@[a-zA-Z0-9_\.]+\.[a-zA-Z]{2,5}", " ", text)
57
58     @staticmethod
59     def remove_urls(text):
60         return re.sub(r"bhttps?:\/\/\S+|www\.\S+", " ", text)
61
62     @staticmethod
63     def remove_html_tags(text):
64         return re.sub(r"<.*>", "", text)
65
66     @staticmethod
67     def clean_text(text):
68         text = TextCleaner.unicode_to_ascii(text.lower().strip())
69         # replacing email addresses with empty string
70         text = TextCleaner.remove_email_addresses(text)
71         # replacing urls with empty string
72         text = TextCleaner.remove_urls(text)
73         # Remove HTML tags
74         text = TextCleaner.remove_html_tags(text)
75         # Expand contraction for eg., wouldn't => would not
76         text = TextCleaner.expand_contractions(text)
77         # creating a space between a word and the punctuation following it
78         text = re.sub(r"([?.!,;])", r" \1 ", text)
79         text = re.sub(r'[" ]+', " ", text)
80         # removes all non-alphabetical characters
81         text = re.sub(r'^a-zA-Z\s+', "", text)
82         # remove extra spaces
83         text = re.sub(" +", " ", text)
84         text = text.strip()
85         return text
86
87
88 class TextPreprocessor:
89     @staticmethod
90     def get_stopwords_pattern():
91         # Stopword list
92         og_stopwords = set(stopwords.words("english"))
93
94         # Define a list of negative words to remove
95         neg_words = ["no", "not", "nor", "neither", "none", "never", "nobody", "nowhere"]
96         custom_stopwords = [word for word in og_stopwords if word not in neg_words]
97         pattern = re.compile(r"\b(" + r"|".join(custom_stopwords) + r")\b\s*")
98         return pattern

```



```

98         return pattern
99
100     @staticmethod
101     def lemmatize_text(text):
102         lemmatizer = WordNetLemmatizer()
103         words = word_tokenize(text)
104         lemmatized_words = [lemmatizer.lemmatize(word) for word in words]
105         return " ".join(lemmatized_words)
106
107     @staticmethod
108     def preprocess_text(text):
109         # replacing all the stopwords
110         text = TextPreprocessor.get_stopwords_pattern().sub("", text)
111         text = TextPreprocessor.lemmatize_text(text)
112         return text
113
114
115 clean_text_vect = np.vectorize(TextCleaner.clean_text)
116 preprocess_text_vect = np.vectorize(TextPreprocessor.preprocess_text)
117
118
119 class DataLoader:
120     @staticmethod
121     def load_data(path):
122         df = pd.read_csv(
123             path,
124             sep="\t",
125             usecols=["review_headline", "review_body", "star_rating"],
126             on_bad_lines="skip",
127             memory_map=True,
128         )
129         return df
130
131
132 class DataProcessor:
133     @staticmethod
134     def filter_columns(df):
135         return df.loc[:, ["review_body", "star_rating"]]
136
137     @staticmethod
138     def convert_star_rating(df):
139         df["star_rating"] = pd.to_numeric(df["star_rating"], errors="coerce")
140         df.dropna(subset=["star_rating"], inplace=True)
141         return df
142
143     @staticmethod
144     def classify_sentiment(df):
145         df["sentiment"] = df["star_rating"].apply(lambda x: 1 if x <= 3 else 2)
146         return df
147
148     @staticmethod
149     def sample_data(df, n_samples, random_state):
150         sampled_df = pd.concat(
151             [
152                 df.query("sentiment==1").sample(n=n_samples, random_state=random_state),
153                 df.query("sentiment==2").sample(n=n_samples, random_state=random_state),
154             ],
155             ignore_index=True,
156         ).sample(frac=1, random_state=random_state)
157
158         sampled_df.drop(columns=["star_rating"], inplace=True)
159         return sampled_df
160
161
162 def clean_and_process_data(path):
163     df = DataLoader.load_data(path)
164     df_filtered = DataProcessor.filter_columns(df)
165     df_filtered = DataProcessor.convert_star_rating(df_filtered)
166     df_filtered = DataProcessor.classify_sentiment(df_filtered)
167
168     balanced_df = DataProcessor.sample_data(
169         df_filtered, Config.N_SAMPLES_EACH_CLASS, Config.RANDOM_STATE
170     )
171
172     balanced_df["review_body"] = balanced_df["review_body"].astype(str)
173
174     # Clean data
175     avg_len_before_clean = balanced_df["review_body"].apply(len).mean()
176     balanced_df["review_body"] = balanced_df["review_body"].apply(clean_text_vect)
177     avg_len_after_clean = balanced_df["review_body"].apply(len).mean()
178
179     # Preprocess data
180     avg_len_before_preprocess = avg_len_after_clean
181     balanced_df["review_body"] = balanced_df["review_body"].apply(preprocess_text_vect)
182     avg_len_after_preprocess = balanced_df["review_body"].apply(len).mean()
183
184     # Print Results
185     print(f"{avg_len_before_clean:.2f}, {avg_len_after_clean:.2f}")
186     print(f"{avg_len_before_preprocess:.2f}, {avg_len_after_preprocess:.2f}")
187
188     return balanced_df
189
190
191 def evaluate_model(model, X_test, y_test):
192     # Predict on the test set
193     y_pred = model.predict(X_test)
194
195     # Calculate evaluation metrics

```

```

196     precision = precision_score(y_test, y_pred, average="binary")
197     recall = recall_score(y_test, y_pred, average="binary")
198     f1 = f1_score(y_test, y_pred, average="binary")
199
200     return precision, recall, f1
201
202
203 def train_evaluate_perceptron(X_train, y_train, X_test, y_test):
204     # Initialize Perceptron model
205     perceptron = Perceptron(max_iter=4000)
206
207     # Train the model
208     perceptron.fit(X_train, y_train)
209
210     # Evaluate model
211     precision, recall, f1 = evaluate_model(perceptron, X_test, y_test)
212     return precision, recall, f1
213
214
215 def train_evaluate_svm(X_train, y_train, X_test, y_test):
216     # Initialize SVM model
217     svm = LinearSVC(max_iter=2500)
218
219     # Train the model
220     svm.fit(X_train, y_train)
221
222     # Evaluate model
223     precision, recall, f1 = evaluate_model(svm, X_test, y_test)
224     return precision, recall, f1
225
226
227 def train_evaluate_logistic_regression(X_train, y_train, X_test, y_test):
228     # Initialize Logistic Regression model
229     log_reg = LogisticRegression(max_iter=4000)
230
231     # Train the model
232     log_reg.fit(X_train, y_train)
233
234     # Evaluate model
235     precision, recall, f1 = evaluate_model(log_reg, X_test, y_test)
236
237     return precision, recall, f1
238
239
240 def train_evaluate_naive_bayes(X_train, y_train, X_test, y_test):
241     # Initialize Naive Bayes model (Multinomial Naive Bayes for text classification)
242     nb_model = MultinomialNB()
243
244     # Train the model
245     nb_model.fit(X_train, y_train)
246
247     # Evaluate model
248     precision, recall, f1 = evaluate_model(nb_model, X_test, y_test)
249
250     return precision, recall, f1
251
252
253 def main():
254     balanced_df = clean_and_process_data(Config.DATA_PATH)
255
256     # Splitting the reviews dataset
257     X_train, X_test, y_train, y_test = train_test_split(
258         balanced_df["review_body"],
259         balanced_df["sentiment"],
260         test_size=Config.TEST_SPLIT,
261         random_state=Config.RANDOM_STATE,
262     )
263
264     # Feature Extraction
265     tfidf_vectorizer = TfidfVectorizer(max_features=Config.NUM_TFIDF_FEATURES)
266     X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
267     X_test_tfidf = tfidf_vectorizer.transform(X_test)
268
269     count_vectorizer = CountVectorizer(max_features=Config.NUM_BOW_FEATURES)
270     X_train_bow = count_vectorizer.fit_transform(X_train)
271     X_test_bow = count_vectorizer.transform(X_test)
272
273     # Train and evaluate Perceptron model using BoW features
274     precision_perceptron_bow, recall_perceptron_bow, f1_perceptron_bow = train_evaluate_perceptron(
275         X_train_bow, y_train, X_test_bow, y_test
276     )
277
278     # Train and evaluate Perceptron model using TF-IDF features
279     (
280         precision_perceptron_tfidf,
281         recall_perceptron_tfidf,
282         f1_perceptron_tfidf,
283     ) = train_evaluate_perceptron(X_train_tfidf, y_train, X_test_tfidf, y_test)
284
285     # Print the results
286     print(f"{precision_perceptron_bow:.4f} {recall_perceptron_bow:.4f} {f1_perceptron_bow:.4f}")
287     print(
288         f"{precision_perceptron_tfidf:.4f} {recall_perceptron_tfidf:.4f} {f1_perceptron_tfidf:.4f}"
289     )
290
291     # Train and evaluate SVM model using BoW features
292     precision_svm_bow, recall_svm_bow, f1_svm_bow = train_evaluate_svm(

```

```

293     X_train_bow, y_train, X_test_bow, y_test
294 )
295
296 # Train and evaluate SVM model using TF-IDF features
297 precision_svm_tfidf, recall_svm_tfidf, f1_svm_tfidf = train_evaluate_svm(
298     X_train_tfidf, y_train, X_test_tfidf, y_test
299 )
300
301 # Print the results
302 print(f"{precision_svm_bow:.4f} {recall_svm_bow:.4f} {f1_svm_bow:.4f}")
303 print(f"{precision_svm_tfidf:.4f} {recall_svm_tfidf:.4f} {f1_svm_tfidf:.4f}")
304
305 # Train and evaluate Logistic Regression model using Bow features
306 precision_lr_bow, recall_lr_bow, f1_lr_bow = train_evaluate_logistic_regression(
307     X_train_bow, y_train, X_test_bow, y_test
308 )
309
310 # Train and evaluate Logistic Regression model using TF-IDF features
311 precision_lr_tfidf, recall_lr_tfidf, f1_lr_tfidf = train_evaluate_logistic_regression(
312     X_train_tfidf, y_train, X_test_tfidf, y_test
313 )
314
315 # Print the results
316 print(f"{precision_lr_bow:.4f} {recall_lr_bow:.4f} {f1_lr_bow:.4f}")
317 print(f"{precision_lr_tfidf:.4f} {recall_lr_tfidf:.4f} {f1_lr_tfidf:.4f}")
318
319 # Train and evaluate Naive Bayes model using Bow features
320 precision_nb_bow, recall_nb_bow, f1_nb_bow = train_evaluate_naive_bayes(
321     X_train_bow, y_train, X_test_bow, y_test
322 )
323
324 # Train and evaluate Naive Bayes model using TF-IDF features
325 precision_nb_tfidf, recall_nb_tfidf, f1_nb_tfidf = train_evaluate_naive_bayes(
326     X_train_tfidf, y_train, X_test_tfidf, y_test
327 )
328
329 # Print the results
330 print(f"{precision_nb_bow:.4f} {recall_nb_bow:.4f} {f1_nb_bow:.4f}")
331 print(f"{precision_nb_tfidf:.4f} {recall_nb_tfidf:.4f} {f1_nb_tfidf:.4f}")
332
333
334 if __name__ == "__main__":
335     main()
336

```

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

```

1  !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