

▼ Dependencies

▼ Install

```
1 !pip install contractions
2 !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

```
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, wordnet
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 nltk.download('averaged_perceptron_tagger')
20
21 import contractions
22
23 from sklearn.model_selection import train_test_split
24 from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
25 from sklearn.metrics import precision_score, recall_score, f1_score
26
27 from sklearn.linear_model import Perceptron, LogisticRegression
28 from sklearn.svm import SVC, LinearSVC
29 from sklearn.naive_bayes import MultinomialNB
30
31 from sklearn.experimental import enable_halving_search_cv
32 from sklearn.model_selection import HalvingGridSearchCV, GridSearchCV
33
34 %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

```
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 = 8000

time: 903 µs (started: 2023-09-18 04:58:29 +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: 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.

```
1 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

```
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: 831 ms (started: 2023-09-18 04:58:30 +00:00)

1 del df_small

time: 661 µs (started: 2023-09-18 04:58:30 +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: 49.9 s (started: 2023-09-18 04:58:30 +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 &#34;new&#34; 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.

```
1 # Check incosistencies 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: 175 ms (started: 2023-09-18 04:59:20 +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: 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

```
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	
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

```
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: 965 µs (started: 2023-09-18 04:59:24 +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 # Drop rows with empty review_body
7 balanced_df = balanced_df.loc[balanced_df["review_body"].str.strip() != ""]
8
9 # Calculate average length of reviews after cleaning
10 avg_len_after_clean = balanced_df["review_body"].apply(len).mean()
11
12 print(f'Avg. Length of Reviews Before Cleaning: {avg_len_before_clean:.2f} characters')
13 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

```
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 lemmatizer = WordNetLemmatizer()
10
11 def pos_tagger(nltk_tag):
12     if nltk_tag.startswith('J'):
13         return wordnet.ADJ
14     elif nltk_tag.startswith('V'):
15         return wordnet.VERB
16     elif nltk_tag.startswith('N'):
17         return wordnet.NOUN
18     elif nltk_tag.startswith('R'):
19         return wordnet.ADV
20     else:
21         return None
22
23 def lemmatize_text_with_pos_tagging(text):
24     words = nltk.pos_tag(word_tokenize(text))
25     words = map(lambda x: (x[0], pos_tagger(x[1])), words)
26     lemmatized_words = [
27         lemmatizer.lemmatize(word, tag)
28         if tag else word
29         for word, tag in words
30     ]
31     return ' '.join(lemmatized_words)
32
33 def lemmatize_text(text):
34     words = word_tokenize(text)
35     lemmatized_words = [lemmatizer.lemmatize(word) for word in words]
36     return ' '.join(lemmatized_words)
37
38 def preprocess_text(text):
39     # replacing all the stopwords
40     text = pattern.sub('',text)
41     text = lemmatize_text(text)
42     return text
43
44 def preprocess_text_with_pos_tagging(text):
45     # replacing all the stopwords
46     text = pattern.sub('',text)
47     text = lemmatize_text_with_pos_tagging(text)
48     return text
49
50 preprocess_text_vect = np.vectorize(preprocess_text)
51 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

```
1 # Calculate average length of reviews before cleaning
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 # Drop rows with empty review_body
7 balanced_df = balanced_df.loc[balanced_df["review_body"].str.strip() != ""]
8
9 # Calculate average length of reviews after cleaning
10 avg_len_after_preprocess = balanced_df["review_body"].apply(len).mean()
11
12 print(f'Avg. Length of Reviews Before Preprocessing: {avg_len_before_preprocess:.2f} characters')
13 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

```
1 balanced_df_pos_tag = balanced_df.copy(deep=True)
2 balanced_df_pos_tag["review_body"] = balanced_df_pos_tag["review_body"].apply(preprocess_text_vect_pos_tag)
3
4 # Drop rows with empty review_body
5 balanced_df_pos_tag = balanced_df_pos_tag.loc[balanced_df["review_body"].str.strip() != ""]
6
7 # Calculate average length of reviews after cleaning
8 avg_len_after_preprocess_pos_tag = balanced_df_pos_tag["review_body"].apply(len).mean()
9
10 print(f'Avg. Length of Reviews Before Preprocessing: {avg_len_before_preprocess:.2f} characters')
11 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

```
1 # Without Pos Tagging lem
2 X_train, X_test, y_train, y_test = train_test_split(
3     balanced_df['review_body'],
4     balanced_df['sentiment'],
5     test_size=Config.TEST_SPLIT,
6     random_state=Config.RANDOM_STATE
7 )
8
9 # With Pos tagging lem
10 X_train_pt, X_test_pt, y_train_pt, y_test_pt = train_test_split(
11     balanced_df_pos_tag['review_body'],
12     balanced_df_pos_tag['sentiment'],
13     test_size=Config.TEST_SPLIT,
14     random_state=Config.RANDOM_STATE
15 )
```

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

▼ Feature Extraction

▼ TF-IDF

```
1 # Without Pos Tagging lem
2 tfidf_vectorizer = TfidfVectorizer(max_features=Config.NUM_TFIDF_FEATURES)
3 X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
4 X_test_tfidf = tfidf_vectorizer.transform(X_test)
5
6 # With Pos tagging lem
7 tfidf_vectorizer_pt = TfidfVectorizer(max_features=Config.NUM_TFIDF_FEATURES)
8 X_train_tfidf_pt = tfidf_vectorizer_pt.fit_transform(X_train_pt)
9 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

```
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)
4
5 count_vectorizer_pt = CountVectorizer(max_features=Config.NUM_BOW_FEATURES)
6 X_train_bow_pt = count_vectorizer_pt.fit_transform(X_train_pt)
7 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

```
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
11
12
13 def train_and_evaluate_model(model_class, X_train, y_train, X_test, y_test, **model_params):
14     # Initialize model
15     model = model_class(**model_params)
16
17     # Train the model
18     model.fit(X_train, y_train)
19
20     # Evaluate model
```

```
21 precision, recall, f1 = evaluate_model(model, X_test, y_test)
22
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

```
1 # Define the parameter grid to search
2 param_grid = {
3     'max_iter': [1000, 2000, 4000, 8000, 10000, 12000],
4     'tol': [1e-1, 1e-2, 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=5, # Number of cross-validation folds
17     n_jobs=-1,
18     verbose=2
19 )
20
21 grid_search_bow.fit(X_train_bow, y_train)
22
23 # Get the best parameters and model for BoW
24 best_params_bow = grid_search_bow.best_params_
25 best_model_bow = grid_search_bow.best_estimator_
26
27 # Evaluate the best model for BoW
28 precision_perceptron_bow, recall_perceptron_bow, f1_perceptron_bow = evaluate_model(best_model_bow, X_test_bow, y_test)
29
30 # Print the results for BoW
31 print(f'Best Parameters (Bow): {best_params_bow}')
32 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)
```

```
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=5, # Number of cross-validation folds
7     n_jobs=-1,
8     verbose=4
9 )
10
11 grid_search_tfidf.fit(X_train_tfidf, y_train)
12
13 # Get the best parameters and model for TF-IDF
14 best_params_tfidf = grid_search_tfidf.best_params_
15 best_model_tfidf = grid_search_tfidf.best_estimator_
16
17 # Evaluate the best model for TF-IDF
18 precision_perceptron_tfidf, recall_perceptron_tfidf, f1_perceptron_tfidf = evaluate_model(best_model_tfidf, X_test_tfidf, y_test)
19
20 # Print the results for TF-IDF
21 print(f'Best Parameters (TF-IDF): {best_params_tfidf}')
22 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

```
1 # Train and evaluate Perceptron model using BoW features
2 precision_perceptron_bow, recall_perceptron_bow, f1_perceptron_bow = train_and_evaluate_model(
3     Perceptron,
4     X_train_bow, y_train, X_test_bow, y_test,
5     max_iter=1000
6 )
7
8 # Train and evaluate Perceptron model using TF-IDF features
9 precision_perceptron_tfidf, recall_perceptron_tfidf, f1_perceptron_tfidf = train_and_evaluate_model(
10     Perceptron,
11     X_train_tfidf, y_train, X_test_tfidf, y_test,
12     max_iter=1000
13 )
14
15 # Print the results
16 print(f'Precision Recall F1 (Bow): {precision_perceptron_bow:.4f} {recall_perceptron_bow:.4f} {f1_perceptron_bow:.4f}')
17 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

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

```
6 )
7
8 # Train and evaluate Perceptron model using TF-IDF features
9 precision_perceptron_tfidf_pt, recall_perceptron_tfidf_pt, f1_perceptron_tfidf_pt = train_and_evaluate_model(
10     Perceptron,
11     X_train_tfidf_pt, y_train, X_test_tfidf_pt, y_test,
12     max_iter=1000
13 )
14
15 # Print the results
16 print(f'Precision Recall F1 (BoW): {precision_perceptron_bow_pt:.4f} {recall_perceptron_bow_pt:.4f} {f1_perceptron_bow_pt:.4f}')
17 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

```
1 # Train and evaluate SVM model using BoW features
2 precision_svm_bow, recall_svm_bow, f1_svm_bow = train_and_evaluate_model(
3     LinearSVC,
4     X_train_bow, y_train, X_test_bow, y_test,
5     max_iter=2000
6 )
7
8 # Train and evaluate SVM model using TF-IDF features
9 precision_svm_tfidf, recall_svm_tfidf, f1_svm_tfidf = train_and_evaluate_model(
10     LinearSVC,
11     X_train_tfidf, y_train, X_test_tfidf, y_test,
12     max_iter=2000
13 )
14
15 # Print the results
16 print(f'Precision Recall F1 (SVM, BoW): {precision_svm_bow:.4f} {recall_svm_bow:.4f} {f1_svm_bow:.4f}')
17 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

```
1 # Train and evaluate SVM model using BoW features
2 precision_svm_bow, recall_svm_bow, f1_svm_bow = train_and_evaluate_model(
3     LinearSVC,
4     X_train_bow_pt, y_train, X_test_bow_pt, y_test,
5     max_iter=2000
6 )
7
8 # Train and evaluate SVM model using TF-IDF features
9 precision_svm_tfidf, recall_svm_tfidf, f1_svm_tfidf = train_and_evaluate_model(
10     LinearSVC,
11     X_train_tfidf_pt, y_train, X_test_tfidf_pt, y_test,
12     max_iter=2000
13 )
14
15 # Print the results
16 print(f'Precision Recall F1 (SVM, BoW): {precision_svm_bow:.4f} {recall_svm_bow:.4f} {f1_svm_bow:.4f}')
17 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

```
1 # Train and evaluate Logistic Regression model using BoW features
2 precision_lr_bow, recall_lr_bow, f1_lr_bow = train_and_evaluate_model(
3     LogisticRegression,
4     X_train_bow, y_train, X_test_bow, y_test,
5     max_iter=2000
6 )
7
8 # Train and evaluate Logistic Regression model using TF-IDF features
9 precision_lr_tfidf, recall_lr_tfidf, f1_lr_tfidf = train_and_evaluate_model(
10     LogisticRegression,
11     X_train_tfidf, y_train, X_test_tfidf, y_test,
12     max_iter=2000
13 )
14
15 # Print the results
16 print(f'Precision Recall F1 (Logistic Regression, BoW): {precision_lr_bow:.4f} {recall_lr_bow:.4f} {f1_lr_bow:.4f}')
17 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

```
1 # Train and evaluate Logistic Regression model using BoW features
2 precision_lr_bow, recall_lr_bow, f1_lr_bow = train_and_evaluate_model(
3     LogisticRegression,
4     X_train_bow_pt, y_train, X_test_bow_pt, y_test,
5     max_iter=2000
6 )
7
8 # Train and evaluate Logistic Regression model using TF-IDF features
9 precision_lr_tfidf, recall_lr_tfidf, f1_lr_tfidf = train_and_evaluate_model(
10     LogisticRegression,
11     X_train_tfidf_pt, y_train, X_test_tfidf_pt, y_test,
12     max_iter=2000
```

```
13 )
14
15 # Print the results
16 print(f'Precision Recall F1 (Logistic Regression, BoW): {precision_lr_bow:.4f} {recall_lr_bow:.4f} {f1_lr_bow:.4f}')
```

```
17 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

```
1 # Train and evaluate Naive Bayes model using Bow features
2 precision_nb_bow, recall_nb_bow, f1_nb_bow = train_and_evaluate_model(
3     MultinomialNB,
4     X_train_bow, y_train, X_test_bow, y_test
5 )
6
7 # Train and evaluate Naive Bayes model using TF-IDF features
8 precision_nb_tfidf, recall_nb_tfidf, f1_nb_tfidf = train_and_evaluate_model(
9     MultinomialNB,
10    X_train_tfidf, y_train, X_test_tfidf, y_test
11 )
12
13 # Print the results
14 print(f'Precision Recall F1 (Naive Bayes, BoW): {precision_nb_bow:.4f} {recall_nb_bow:.4f} {f1_nb_bow:.4f}')
```

```
15 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

```
1 # Train and evaluate Naive Bayes model using Bow features
2 precision_nb_bow, recall_nb_bow, f1_nb_bow = train_and_evaluate_model(
3     MultinomialNB,
4     X_train_bow_pt, y_train, X_test_bow_pt, y_test
5 )
6
7 # Train and evaluate Naive Bayes model using TF-IDF features
8 precision_nb_tfidf, recall_nb_tfidf, f1_nb_tfidf = train_and_evaluate_model(
9     MultinomialNB,
10    X_train_tfidf_pt, y_train, X_test_tfidf_pt, y_test
11 )
12
13 # Print the results
14 print(f'Precision Recall F1 (Naive Bayes, BoW): {precision_nb_bow:.4f} {recall_nb_bow:.4f} {f1_nb_bow:.4f}')
```

```
15 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

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

```

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

▼ With negative stopwords

```
1 %%writefile HW1-CSCI544-w-neg-sw.py
2 # Python Version: 3.10.12
3
4 # Python Version: 3.10.12
5
6 import re
7 import unicodedata
8
9 import warnings
10
11 warnings.filterwarnings("ignore")
12
13 import numpy as np
14 import pandas as pd
15
16 import nltk
17 from nltk.corpus import stopwords, wordnet
18 from nltk.stem import WordNetLemmatizer
19 from nltk.tokenize import word_tokenize
20
21 nltk.download("punkt", quiet=True)
22 nltk.download("wordnet", quiet=True)
23 nltk.download("stopwords", quiet=True)
24 nltk.download("averaged_perceptron_tagger", quiet=True)
25
26 import contractions
27
28 from sklearn.model_selection import train_test_split
29 from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
30 from sklearn.metrics import precision_score, recall_score, f1_score
31
32 from sklearn.linear_model import Perceptron, LogisticRegression
33 from sklearn.svm import LinearSVC
34 from sklearn.naive_bayes import MultinomialNB
35
36
37 class Config:
38     RANDOM_STATE = 56
39     DATA_PATH = "amazon_reviews_us_Office_Products_v1_00.tsv.gz"
40     TEST_SPLIT = 0.2
41     N_SAMPLES_EACH_CLASS = 50000
42     NUM_TFIDF_FEATURES = 5000
43     NUM_BOW_FEATURES = 5000
```



```

44
45
46 class DataLoader:
47     @staticmethod
48     def load_data(path):
49         df = pd.read_csv(
50             path,
51             sep="\t",
52             usecols=["review_headline", "review_body", "star_rating"],
53             on_bad_lines="skip",
54             memory_map=True,
55         )
56         return df
57
58
59 class DataProcessor:
60     @staticmethod
61     def filter_columns(df):
62         return df.loc[:, ["review_body", "star_rating"]]
63
64     @staticmethod
65     def convert_star_rating(df):
66         df["star_rating"] = pd.to_numeric(df["star_rating"], errors="coerce")
67         df.dropna(subset=["star_rating"], inplace=True)
68         return df
69
70     @staticmethod
71     def classify_sentiment(df):
72         df["sentiment"] = df["star_rating"].apply(lambda x: 1 if x <= 3 else 2)
73         return df
74
75     @staticmethod
76     def sample_data(df, n_samples, random_state):
77         sampled_df = pd.concat(
78             [
79                 df.query("sentiment==1").sample(n=n_samples, random_state=random_state),
80                 df.query("sentiment==2").sample(n=n_samples, random_state=random_state),
81             ],
82             ignore_index=True,
83         ).sample(frac=1, random_state=random_state)
84
85         sampled_df.drop(columns=["star_rating"], inplace=True)
86         return sampled_df
87
88
89 class TextCleaner:
90     @staticmethod
91     def unicode_to_ascii(s):
92         return "".join(
93             c for c in unicodedata.normalize("NFD", s) if unicodedata.category(c) != "Mn"
94         )
95
96     @staticmethod
97     def expand_contractions(text):
98         return contractions.fix(text)
99
100     @staticmethod
101     def remove_email_addresses(text):
102         return re.sub(r"[a-zA-Z0-9_\-\.]+@[a-zA-Z0-9_\-\.]+\.[a-zA-Z]{2,5}", " ", text)
103
104     @staticmethod
105     def remove_urls(text):
106         return re.sub(r"\bhttps?:\/\/\/\S+|www\.\S+", " ", text)
107
108     @staticmethod
109     def remove_html_tags(text):
110         return re.sub(r"<.*?>", "", text)
111
112     @staticmethod
113     def clean_text(text):
114         text = TextCleaner.unicode_to_ascii(text.lower().strip())
115         # replacing email addresses with empty string
116         text = TextCleaner.remove_email_addresses(text)
117         # replacing urls with empty string
118         text = TextCleaner.remove_urls(text)
119         # Remove HTML tags
120         text = TextCleaner.remove_html_tags(text)
121         # Expand contraction for eg., wouldn't => would not
122         text = TextCleaner.expand_contractions(text)
123         # creating a space between a word and the punctuation following it
124         text = re.sub(r"([?.!,;])", r" \1 ", text)
125         text = re.sub(r'" "', " ", text)
126         # removes all non-alphabetical characters
127         text = re.sub(r"[^a-zA-Z\s]", "", text)
128         # remove extra spaces
129         text = re.sub(" +", " ", text)
130         text = text.strip()
131         return text
132
133
134 class TextPreprocessor:
135     lemmatizer = WordNetLemmatizer()
136
137     @staticmethod
138     def get_stopwords_pattern():
139         # Stopword list
140         og_stopwords = set(stopwords.words("english"))
141
142         # Define a list of negative words to remove
143         neg_words = ["no", "not", "nor", "neither", "none", "never", "nobody", "nowhere"]
144         # custom_stopwords = [word for word in og_stopwords if word not in neg_words]
145         pattern = re.compile(r"\b(" + r"|".join(og_stopwords) + r")\b\s*")
146         return pattern
147
148     @staticmethod
149     def pos_tagger(tag):
150         if tag.startswith("J"):
151             return wordnet.ADJ
152         elif tag.startswith("V"):
153             return wordnet.VERB
154         elif tag.startswith("N"):
155             return wordnet.NOUN
156         elif tag.startswith("R"):
157             return wordnet.ADV
158         else:
159             return None
160
161     @staticmethod
162     def lemmatize_text_using_pos_tags(text):

```

```

163     words = nltk.pos_tag(word_tokenize(text))
164     words = map(lambda x: (x[0], TextPreprocessor.pos_tagger(x[1])), words)
165     lemmatized_words = [
166         TextPreprocessor.lemmatizer.lemmatize(word, tag) if tag else word for word, tag in words
167     ]
168     return " ".join(lemmatized_words)
169
170 @staticmethod
171 def lemmatize_text(text):
172     words = word_tokenize(text)
173     lemmatized_words = [TextPreprocessor.lemmatizer.lemmatize(word) for word in words]
174     return " ".join(lemmatized_words)
175
176 pattern = get_stopwords_pattern()
177
178 @staticmethod
179 def preprocess_text(text):
180     # replacing all the stopwords
181     text = TextPreprocessor.pattern.sub("", text)
182     text = TextPreprocessor.lemmatize_text(text)
183     return text
184
185
186 clean_text_vect = np.vectorize(TextCleaner.clean_text)
187 preprocess_text_vect = np.vectorize(TextPreprocessor.preprocess_text)
188
189
190 def clean_and_process_data(path):
191     df = DataLoader.load_data(path)
192     df_filtered = DataProcessor.filter_columns(df)
193     df_filtered = DataProcessor.convert_star_rating(df_filtered)
194     df_filtered = DataProcessor.classify_sentiment(df_filtered)
195
196     balanced_df = DataProcessor.sample_data(
197         df_filtered, Config.N_SAMPLES_EACH_CLASS, Config.RANDOM_STATE
198     )
199
200     balanced_df["review_body"] = balanced_df["review_body"].astype(str)
201
202     # Clean data
203     avg_len_before_clean = balanced_df["review_body"].apply(len).mean()
204     balanced_df["review_body"] = balanced_df["review_body"].apply(clean_text_vect)
205     # Drop reviews that are empty
206     balanced_df = balanced_df.loc[balanced_df["review_body"].str.strip() != ""]
207     avg_len_after_clean = balanced_df["review_body"].apply(len).mean()
208
209     # Preprocess data
210     avg_len_before_preprocess = avg_len_after_clean
211     balanced_df["review_body"] = balanced_df["review_body"].apply(preprocess_text_vect)
212     avg_len_after_preprocess = balanced_df["review_body"].apply(len).mean()
213
214     # Print Results
215     print(f"{avg_len_before_clean:.2f}, {avg_len_after_clean:.2f}")
216     print(f"{avg_len_before_preprocess:.2f}, {avg_len_after_preprocess:.2f}")
217
218     return balanced_df
219
220
221 def evaluate_model(model, X_test, y_test):
222     # Predict on the test set
223     y_pred = model.predict(X_test)
224
225     # Calculate evaluation metrics
226     precision = precision_score(y_test, y_pred, average="binary")
227     recall = recall_score(y_test, y_pred, average="binary")
228     f1 = f1_score(y_test, y_pred, average="binary")
229
230     return precision, recall, f1
231
232
233 def train_and_evaluate_model(model_class, X_train, y_train, X_test, y_test, **model_params):
234     # Initialize model
235     model = model_class(**model_params)
236
237     # Train the model
238     model.fit(X_train, y_train)
239
240     # Evaluate model
241     precision, recall, f1 = evaluate_model(model, X_test, y_test)
242     return model, precision, recall, f1
243
244
245 def main():
246     balanced_df = clean_and_process_data(Config.DATA_PATH)
247
248     # Splitting the reviews dataset
249     X_train, X_test, y_train, y_test = train_test_split(
250         balanced_df["review_body"],
251         balanced_df["sentiment"],
252         test_size=Config.TEST_SPLIT,
253         random_state=Config.RANDOM_STATE,
254     )
255
256     # Feature Extraction
257     tfidf_vectorizer = TfidfVectorizer(max_features=Config.NUM_TFIDF_FEATURES)
258     X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
259     X_test_tfidf = tfidf_vectorizer.transform(X_test)
260
261     count_vectorizer = CountVectorizer(max_features=Config.NUM_BOW_FEATURES)
262     X_train_bow = count_vectorizer.fit_transform(X_train)
263     X_test_bow = count_vectorizer.transform(X_test)
264
265     # Train and evaluate Perceptron model using Bow features
266     (
267         _,
268         precision_perceptron_bow,
269         recall_perceptron_bow,
270         f1_perceptron_bow,
271     ) = train_and_evaluate_model(
272         Perceptron, X_train_bow, y_train, X_test_bow, y_test, max_iter=4000
273     )
274
275     # Train and evaluate Perceptron model using TF-IDF features
276     (
277         _,
278         precision_perceptron_tfidf,
279         recall_perceptron_tfidf,
280         f1_perceptron_tfidf,
281     ) = train_and_evaluate_model(

```

```
282     Perceptron, X_train_tfidf, y_train, X_test_tfidf, y_test, max_iter=4000
283 )
284
285 # Train and evaluate SVM model using Bow features
286 _, precision_svm_bow, recall_svm_bow, f1_svm_bow = train_and_evaluate_model(
287     LinearSVC, X_train_bow, y_train, X_test_bow, y_test, max_iter=2500
288 )
289
290 # Train and evaluate SVM model using TF-IDF features
291 _, precision_svm_tfidf, recall_svm_tfidf, f1_svm_tfidf = train_and_evaluate_model(
292     LinearSVC, X_train_tfidf, y_train, X_test_tfidf, y_test, max_iter=2500
293 )
294
295 # Train and evaluate Logistic Regression model using Bow features
296 _, precision_lr_bow, recall_lr_bow, f1_lr_bow = train_and_evaluate_model(
297     LogisticRegression, X_train_bow, y_train, X_test_bow, y_test, max_iter=4000
298 )
299
300 # Train and evaluate Logistic Regression model using TF-IDF features
301 _, precision_lr_tfidf, recall_lr_tfidf, f1_lr_tfidf = train_and_evaluate_model(
302     LogisticRegression, X_train_tfidf, y_train, X_test_tfidf, y_test, max_iter=4000
303 )
304
305 # Train and evaluate Naive Bayes model using Bow features
306 _, precision_nb_bow, recall_nb_bow, f1_nb_bow = train_and_evaluate_model(
307     MultinomialNB, X_train_bow, y_train, X_test_bow, y_test
308 )
309
310 # Train and evaluate Naive Bayes model using TF-IDF features
311 _, precision_nb_tfidf, recall_nb_tfidf, f1_nb_tfidf = train_and_evaluate_model(
312     MultinomialNB, X_train_tfidf, y_train, X_test_tfidf, y_test
313 )
314
315 # Print the results
316 print(f"{precision_perceptron_bow:.4f} {recall_perceptron_bow:.4f} {f1_perceptron_bow:.4f}")
317 print(
318     f"{precision_perceptron_tfidf:.4f} {recall_perceptron_tfidf:.4f} {f1_perceptron_tfidf:.4f}"
319 )
320
321 print(f"{precision_svm_bow:.4f} {recall_svm_bow:.4f} {f1_svm_bow:.4f}")
322 print(f"{precision_svm_tfidf:.4f} {recall_svm_tfidf:.4f} {f1_svm_tfidf:.4f}")
323
324 print(f"{precision_lr_bow:.4f} {recall_lr_bow:.4f} {f1_lr_bow:.4f}")
325 print(f"{precision_lr_tfidf:.4f} {recall_lr_tfidf:.4f} {f1_lr_tfidf:.4f}")
326
327 print(f"{precision_nb_bow:.4f} {recall_nb_bow:.4f} {f1_nb_bow:.4f}")
328 print(f"{precision_nb_tfidf:.4f} {recall_nb_tfidf:.4f} {f1_nb_tfidf:.4f}")
329
330
331 if __name__ == "__main__":
332     main()
333
```

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

▼ Without negative stopwords

```
1  %%writefile HW1-CSCI544-wo-neg-sw.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, wordnet
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 nltk.download("averaged_perceptron_tagger", quiet=True)
23
24 import contractions
25
26 from sklearn.model_selection import train_test_split
27 from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
28 from sklearn.metrics import precision_score, recall_score, f1_score
29
30 from sklearn.linear_model import Perceptron, LogisticRegression
31 from sklearn.svm import LinearSVC
32 from sklearn.naive_bayes import MultinomialNB
33
34
35 class Config:
36     RANDOM_STATE = 56
37     DATA_PATH = "amazon_reviews_us_Office_Products_v1_00.tsv.gz"
38     TEST_SPLIT = 0.2
39     N_SAMPLES_EACH_CLASS = 50000
40     NUM_TFIDF_FEATURES = 5000
41     NUM_BOW_FEATURES = 5000
42
43
44 class DataLoader:
45     @staticmethod
46     def load_data(path):
47         df = pd.read_csv(
48             path,
49             sep="\t",
50             usecols=["review_headline", "review_body", "star_rating"],
51             on_bad_lines="skip",
52             memory_map=True,
53         )
54         return df
55
56
57 class DataProcessor:
58     @staticmethod
59     def filter_columns(df):
60         return df.loc[:, ["review_body", "star_rating"]]
```

```

61
62     @staticmethod
63     def convert_star_rating(df):
64         df["star_rating"] = pd.to_numeric(df["star_rating"], errors="coerce")
65         df.dropna(subset=["star_rating"], inplace=True)
66         return df
67
68     @staticmethod
69     def classify_sentiment(df):
70         df["sentiment"] = df["star_rating"].apply(lambda x: 1 if x <= 3 else 2)
71         return df
72
73     @staticmethod
74     def sample_data(df, n_samples, random_state):
75         sampled_df = pd.concat(
76             [
77                 df.query("sentiment==1").sample(n=n_samples, random_state=random_state),
78                 df.query("sentiment==2").sample(n=n_samples, random_state=random_state),
79             ],
80             ignore_index=True,
81         ).sample(frac=1, random_state=random_state)
82
83         sampled_df.drop(columns=["star_rating"], inplace=True)
84         return sampled_df
85
86
87 class TextCleaner:
88     @staticmethod
89     def unicode_to_ascii(s):
90         return "".join(
91             c for c in unicodedata.normalize("NFD", s) if unicodedata.category(c) != "Mn"
92         )
93
94     @staticmethod
95     def expand_contractions(text):
96         return contractions.fix(text)
97
98     @staticmethod
99     def remove_email_addresses(text):
100         return re.sub(r"[a-zA-Z0-9_\-\.]+@[a-zA-Z0-9_\-\.]+\.[a-zA-Z]{2,5}", " ", text)
101
102     @staticmethod
103     def remove_urls(text):
104         return re.sub(r"\bhttps?:\/\/\/\S+|www\.\S+", " ", text)
105
106     @staticmethod
107     def remove_html_tags(text):
108         return re.sub(r"<.*?>", "", text)
109
110     @staticmethod
111     def clean_text(text):
112         text = TextCleaner.unicode_to_ascii(text.lower().strip())
113         # replacing email addresses with empty string
114         text = TextCleaner.remove_email_addresses(text)
115         # replacing urls with empty string
116         text = TextCleaner.remove_urls(text)
117         # Remove HTML tags
118         text = TextCleaner.remove_html_tags(text)
119         # Expand contraction for eg., wouldn't => would not
120         text = TextCleaner.expand_contractions(text)
121         # creating a space between a word and the punctuation following it
122         text = re.sub(r"([?.!,;])", r" \1 ", text)
123         text = re.sub(r'[" ]+', " ", text)
124         # removes all non-alphabetical characters
125         text = re.sub(r"^[a-zA-Z\s]+", "", text)
126         # remove extra spaces
127         text = re.sub(" +", " ", text)
128         text = text.strip()
129         return text
130
131
132 class TextPreprocessor:
133     lemmatizer = WordNetLemmatizer()
134
135     @staticmethod
136     def get_stopwords_pattern():
137         # Stopword list
138         og_stopwords = set(stopwords.words("english"))
139
140         # Define a list of negative words to remove
141         neg_words = ["no", "not", "nor", "neither", "none", "never", "nobody", "nowhere"]
142         custom_stopwords = [word for word in og_stopwords if word not in neg_words]
143         pattern = re.compile(r"\b(" + r"|".join(custom_stopwords) + r")\b\s*")
144         return pattern
145
146     @staticmethod
147     def pos_tagger(tag):
148         if tag.startswith("J"):
149             return wordnet.ADJ
150         elif tag.startswith("V"):
151             return wordnet.VERB
152         elif tag.startswith("N"):
153             return wordnet.NOUN
154         elif tag.startswith("R"):
155             return wordnet.ADV
156         else:
157             return None
158
159     @staticmethod
160     def lemmatize_text_using_pos_tags(text):
161         words = nltk.pos_tag(word_tokenize(text))
162         words = map(lambda x: (x[0], TextPreprocessor.pos_tagger(x[1])), words)
163         lemmatized_words = [
164             TextPreprocessor.lemmatizer.lemmatize(word, tag) if tag else word for word, tag in words
165         ]
166         return " ".join(lemmatized_words)
167
168     @staticmethod
169     def lemmatize_text(text):
170         words = word_tokenize(text)
171         lemmatized_words = [TextPreprocessor.lemmatizer.lemmatize(word) for word in words]
172         return " ".join(lemmatized_words)
173
174     pattern = get_stopwords_pattern()
175
176     @staticmethod
177     def preprocess_text(text):
178         # replacing all the stopwords
179         text = TextPreprocessor.pattern.sub("", text)
180

```

```

180         text = TextPreprocessor.lemmatize_text(text)
181         return text
182
183
184     clean_text_vect = np.vectorize(TextCleaner.clean_text)
185     preprocess_text_vect = np.vectorize(TextPreprocessor.preprocess_text)
186
187
188     def clean_and_process_data(path):
189         df = DataLoader.load_data(path)
190         df_filtered = DataProcessor.filter_columns(df)
191         df_filtered = DataProcessor.convert_star_rating(df_filtered)
192         df_filtered = DataProcessor.classify_sentiment(df_filtered)
193
194         balanced_df = DataProcessor.sample_data(
195             df_filtered, Config.N_SAMPLES_EACH_CLASS, Config.RANDOM_STATE
196         )
197
198         balanced_df["review_body"] = balanced_df["review_body"].astype(str)
199
200         # Clean data
201         avg_len_before_clean = balanced_df["review_body"].apply(len).mean()
202         balanced_df["review_body"] = balanced_df["review_body"].apply(clean_text_vect)
203         # Drop reviews that are empty
204         balanced_df = balanced_df.loc[balanced_df["review_body"].str.strip() != ""]
205         avg_len_after_clean = balanced_df["review_body"].apply(len).mean()
206
207         # Preprocess data
208         avg_len_before_preprocess = avg_len_after_clean
209         balanced_df["review_body"] = balanced_df["review_body"].apply(preprocess_text_vect)
210         avg_len_after_preprocess = balanced_df["review_body"].apply(len).mean()
211
212         # Print Results
213         print(f"{avg_len_before_clean:.2f}, {avg_len_after_clean:.2f}")
214         print(f"{avg_len_before_preprocess:.2f}, {avg_len_after_preprocess:.2f}")
215
216         return balanced_df
217
218
219     def evaluate_model(model, X_test, y_test):
220         # Predict on the test set
221         y_pred = model.predict(X_test)
222
223         # Calculate evaluation metrics
224         precision = precision_score(y_test, y_pred, average="binary")
225         recall = recall_score(y_test, y_pred, average="binary")
226         f1 = f1_score(y_test, y_pred, average="binary")
227
228         return precision, recall, f1
229
230
231     def train_and_evaluate_model(model_class, X_train, y_train, X_test, y_test, **model_params):
232         # Initialize model
233         model = model_class(**model_params)
234
235         # Train the model
236         model.fit(X_train, y_train)
237
238         # Evaluate model
239         precision, recall, f1 = evaluate_model(model, X_test, y_test)
240         return model, precision, recall, f1
241
242
243     def main():
244         balanced_df = clean_and_process_data(Config.DATA_PATH)
245
246         # Splitting the reviews dataset
247         X_train, X_test, y_train, y_test = train_test_split(
248             balanced_df["review_body"],
249             balanced_df["sentiment"],
250             test_size=Config.TEST_SPLIT,
251             random_state=Config.RANDOM_STATE,
252         )
253
254         # Feature Extraction
255         tfidf_vectorizer = TfidfVectorizer(max_features=Config.NUM_TFIDF_FEATURES)
256         X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
257         X_test_tfidf = tfidf_vectorizer.transform(X_test)
258
259         count_vectorizer = CountVectorizer(max_features=Config.NUM_BOW_FEATURES)
260         X_train_bow = count_vectorizer.fit_transform(X_train)
261         X_test_bow = count_vectorizer.transform(X_test)
262
263         # Train and evaluate Perceptron model using Bow features
264         (
265             _,
266             precision_perceptron_bow,
267             recall_perceptron_bow,
268             f1_perceptron_bow,
269         ) = train_and_evaluate_model(
270             Perceptron, X_train_bow, y_train, X_test_bow, y_test, max_iter=4000
271         )
272
273         # Train and evaluate Perceptron model using TF-IDF features
274         (
275             _,
276             precision_perceptron_tfidf,
277             recall_perceptron_tfidf,
278             f1_perceptron_tfidf,
279         ) = train_and_evaluate_model(
280             Perceptron, X_train_tfidf, y_train, X_test_tfidf, y_test, max_iter=4000
281         )
282
283         # Train and evaluate SVM model using Bow features
284         _, precision_svm_bow, recall_svm_bow, f1_svm_bow = train_and_evaluate_model(
285             LinearSVC, X_train_bow, y_train, X_test_bow, y_test, max_iter=2500
286         )
287
288         # Train and evaluate SVM model using TF-IDF features
289         _, precision_svm_tfidf, recall_svm_tfidf, f1_svm_tfidf = train_and_evaluate_model(
290             LinearSVC, X_train_tfidf, y_train, X_test_tfidf, y_test, max_iter=2500
291         )
292
293         # Train and evaluate Logistic Regression model using Bow features
294         _, precision_lr_bow, recall_lr_bow, f1_lr_bow = train_and_evaluate_model(
295             LogisticRegression, X_train_bow, y_train, X_test_bow, y_test, max_iter=4000
296         )
297
298         # Train and evaluate Logistic Regression model using TF-IDF features
299         _, precision_lr_tfidf, recall_lr_tfidf, f1_lr_tfidf = train_and_evaluate_model(

```

```
299 _, precision_lr_tfidf, recall_lr_tfidf, f1_lr_tfidf = train_and_evaluate_model(
300     LogisticRegression, X_train_tfidf, y_train, X_test_tfidf, y_test, max_iter=4000
301 )
302
303 # Train and evaluate Naive Bayes model using BoW features
304 _, precision_nb_bow, recall_nb_bow, f1_nb_bow = train_and_evaluate_model(
305     MultinomialNB, X_train_bow, y_train, X_test_bow, y_test
306 )
307
308 # Train and evaluate Naive Bayes model using TF-IDF features
309 _, precision_nb_tfidf, recall_nb_tfidf, f1_nb_tfidf = train_and_evaluate_model(
310     MultinomialNB, X_train_tfidf, y_train, X_test_tfidf, y_test
311 )
312
313 # Print the results
314 print("{precision_perceptron_bow:.4f} {recall_perceptron_bow:.4f} {f1_perceptron_bow:.4f}")
315 print(
316     f"{precision_perceptron_tfidf:.4f} {recall_perceptron_tfidf:.4f} {f1_perceptron_tfidf:.4f}"
317 )
318
319 print(f"{precision_svm_bow:.4f} {recall_svm_bow:.4f} {f1_svm_bow:.4f}")
320 print(f"{precision_svm_tfidf:.4f} {recall_svm_tfidf:.4f} {f1_svm_tfidf:.4f}")
321
322 print(f"{precision_lr_bow:.4f} {recall_lr_bow:.4f} {f1_lr_bow:.4f}")
323 print(f"{precision_lr_tfidf:.4f} {recall_lr_tfidf:.4f} {f1_lr_tfidf:.4f}")
324
325 print(f"{precision_nb_bow:.4f} {recall_nb_bow:.4f} {f1_nb_bow:.4f}")
326 print(f"{precision_nb_tfidf:.4f} {recall_nb_tfidf:.4f} {f1_nb_tfidf:.4f}")
327
328
329 if __name__ == "__main__":
330     main()
331
```

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

▼ Comparing Results

```
1 !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)

1 !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