CSCI – 544 Applied NLP HW1

Sentiment Analysis with Amazon Reviews Data

Step 1: Dataset Preparation

Download and load the data set into a pandas DataFrame:

Problem faced/learned from:

- Dataset was taking too long to load.

Solutions Tried:

- Checked file delimiters/separators (\t for tsv file)
- Turns out the process was taking too long as it gets stuck trying to load in faulty data as well, so at the end decided to ignore errors and load in rest of the data. The attribute **error_bad_lines = False** of the "read_csv" method helped in this case.

Step 2: Only Keep Reviews and Ratings from the dataset

Separated only the "review_body" and "star_rating" columns.

Stats:

- A total of 4874890 entries were read. Out of this 246 "review_body" were null/empty and were dropped.

```
reviews.info(verbose = True,show_counts = True)
print(reviews.isnull().sum())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4874890 entries, 0 to 4874889
Data columns (total 2 columns):
 #   Column       Non-Null Count    Dtype
---  ------       --------------    -----
 0   review_body  4874644 non-null  object
 1   star_rating  4874887 non-null  float64
dtypes: float64(1), object(1)
memory usage: 74.4+ MB
review_body    246
star_rating      3
```

- The input data is heavily skewed towards the rating 5.

```
reviews_cpy['star_rating'].value_counts()
```

```
5.0    3124595
4.0     731701
1.0     426870
3.0     349539
2.0     241939
Name: star_rating, dtype: int64
```

Sample Reviews:

Three Random Review samples are provided in the screenshot below with each a rating of 4, 5 & 5 respectively.

```
# Get 3 random rows
reviews_cpy.sample(3)
```

	review_body	star_rating	ratings
1314152	Solid and easy!	4.0	1
2241406	These worked well for the desserts of our scho...	5.0	1
2362029	works well	5.0	1

Step 3: Labelling Reviews:

We label data points based on the criteria that ratings less than or equal to 2 are 0(negative case) and greater than 3 are labelled 1(positive case).

```
# 0 is negative sentiment classes
# 1 is positive sentiment classes
#Class 3 or dropped reviews count here
print('Class 3 or dropped  reviews count : ',len(reviews_cpy['label']) - len(rev
print('Class 0 or negative reviews count : ',reviews['label'].value_counts()[0])
print('Class 1 or positive reviews count : ',reviews['label'].value_counts()[1])
```

```
Class 3 or dropped  reviews count :  349539
Class 0 or negative reviews count :  426870
Class 1 or positive reviews count :  4098235
```

As can be seen above the class wise count of reviews is:
Class 1 (Positive): 4098235

Class 0 (Negative): 426870

Dropped (Neutral): 349539

Step 4: Randomly sample 200,000 reviews with 100,000 from each class respectively

Used ‘shuffle’ and ‘concat’ functions from the pandas library to sample and add 100,000 records from each class

Step 5: Data Cleaning

Average character length of reviews before data cleaning: 325.79(approx.)

```
#Char length of reviews before data cleaning
print('Average character length of reviews :',dataset['review_body'].str.len().s
```

```
Average character length of reviews : 325.785775
```

5a. Converting the reviews to lowercase: Simply used the series str.lower() function

5b. Removing the HTML and URL:

Used Beautiful Soup's html.parser and get_text() functions over the 'review_body' column to remove all the HTML tags from the column.

Used regex to check before and after for columns with <> angular braces for html tags.

After removal there are still some rows with angular brackets however these rows contain angular (<,>) braces in other contexts.

```
#check if any tags remain
[print(x) for x in dataset['review_body'][dataset['review_body'].str.contains('
#only remaining results are lone < or > markers spread around no html tags left
```

```
comes with silicon cord, not metal. < one of the main reasons i bought this w
as for the metal cord.the unit does have an on off switch, i noticed previous
comments stated it didn't. its on the back.i can tell you one thing also, whi
le the probe might be waterproof < its a one piece stainless probe > don't ex
```

5c. Remove all non-alphabetical characters

One of the following steps is to remove contractions however, all contraction words contain an apostrophe symbol. If we remove all non-alphabetical characters in this run, then we won't be able to process them.

Hence, as a workaround**, I've kept only the apostrophe symbol and removed all other non-alphabets from the reviews.**

```
#remove every non alphabet chracter except apostrophe for contractions
dataset['review_body'] = dataset['review_body'].replace("[^a-z ']",'',regex=True
```

After performing the contractions expansion, I'll remove the remaining apostrophe (some are still left as they are not a part of the contractions and could be stray apostrophe).

5d. Remove Contractions

Instead of creating a custom dictionary and replacing the words on my own, I decided to use the contractions library as it boasts a large amount of contraction expansions that would've taken a long time to gather on my own.

Initially checking the reviews for any contractions present. As it can be seen there are some common contractions present like don't or I've etc.

```
#Check for contractions
[print(x) for x in dataset['review_body'][dataset['review_body'].str.contains("[
```

```
i dont't know when did it happen but after using it for two days i found a cr
ack in a blade not strong like steelfortunately amazon refunded full price i
paid
we bought these as a wedding shower gift since we love ours so much  in fact
we own two sets  they are our main mixing and serving bowls lol  and even tho
ugh they aren't made with a microwave in mind they work great for it  we've h
ad one set for  years and the others for  year and going  i truly love how th
ese pour don't skid as you mix and that they look simple and nice
great recipe book and user guide  very helpful i've read every word and i'm s
ure i'll be referring to the recipes for many years jerry
```

After applying the contractions function to the review data, we remove the apostrophe symbol to only have alphabetical characters in our review data.

```
#check if any more apostrophe words remain
[print(x) for x in dataset['review_body'][dataset['review_body'].str.contains("[
# After contractions are removed; delete all the apostrophe as non alphabet char
dataset['review_body'] = dataset['review_body'].replace("[^a-z]",'',regex=True)
```

```
i bought this for my mom on mother's day when i first saw it i was glad to se
e it appeared well built the metal was thick and coated in rubber to prevent
scratches it can be used horizontal or verticaldesigned to be used vertical a
nd both ways help keep things organized my mother is going to install in insi
de a cabinet so it is hidden it allows more space for other items in the cabi
nets
```

We can see there are still some words with apostrophe like "mother's" we need to remove apostrophe from these words.

```
#search for any non alphabetical
[print(x) for x in dataset['review_body'][dataset['review_body'].str.contains("[
```

```
[]
```

5e. Remove extra spaces between the words

For this I used a Regex which replaces more than one whitespace occurring together by only 1 whitespace.

Remove the extra spaces between the words

```
_without_extra_spaces'] = dataset['review_body'].replace('\s+', ' ', regex=True)
```

```
l exist
view_body'][dataset['review_body_without_extra_spaces'].str.contains('\s{2,}')]]
```

```
[]
```

Average character length of reviews after data cleaning: 312.26

```
#Char length of reviews after data cleaning
print('Average character length of reviews after data cleaning:',dataset['review
```

```
Average character length of reviews after data cleaning: 312.26027
```

Step 6: Preprocessing

6a. Removing Stop Words

Used the NLTK library corpus stop words containing the most common stop words and lambda functions to remove the stop words from the review data. We remove stop words to prevent them from acting as bias to the model and as they don't really serve any purpose in determining the sentiment of the review.

6b. Lemmatization

For converting the words to their lemma(roots), I've used the NLTK WordNet Lemmatizer alongside the WhiteSpace Tokenizer to convert the sentence input(review) into tokens(words)

```
dataset.sample(4)
```

_rating	label	review_body_without_spaces	review_body_without_stopwords	review_body_lemmatized
5.0	1	when it says that can be cleaned in the dishwa...	says cleaned dishwasher true thanksgiving cook...	say cleaned dishwasher true thanksgiving cooki...
1.0	0	similar to other reviewers my board cracked in...	similar reviewers board cracked multiple place...	similar reviewer board cracked multiple place ...
5.0	1	best little timer made	best little timer made	best little timer made
5.0	1	nice item	nice item	nice item

Step 7: Feature Extraction

We have multiple ways of extracting feature vectors like Bag of Words, TFIDF etc. Here we are going to use TF-IDF vectorizer to extract features from the preprocessed review data from previous steps.

Made sure to train the vectorizer over the training data only and kept the test data separate to prevent error in results. After training the vectorizer over the training data, it's required to transform both the training and test set before they can be used as features for training and testing the model respectively.

```
print('Train Data Shape:',tfidf_train_data.shape)
print('Test Data Shape:',tfidf_test_data.shape)
```

```
Train Data Shape: (160000, 106963)
Test Data Shape: (40000, 106963)
```

A basic TF-IDF Vectorizer, without any lower or upper limit on term/document frequency outputs 106963 features for each review.

First, we proceed with these features to train the models.

Further test scenarios:

1. Include review title into the review body before all the steps. We initially ignored the column
2. Remove features/terms that only occur 1 time or at most 2 times

Average character length of reviews after data preprocessing: 191.44

```
#Char length of reviews after data preprocessing
print('Average character length of reviews after data preprocessing:',dataset['r
```

```
Average character length of reviews after data preprocessing: 191.44271
```

Step 8 – 11 Train Separate Models and Records the performance of the model on the Train and Test Sets

The table below lists the performance results of the models with the prescribed Data Cleaning and Preprocessing steps:

Model	TRAINING SET (Approximated decimals)				TEST SET (Approximated decimals)			
Name	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
Perceptron	0.901	0.925	0.873	0.898	0.824	0.851	0.786	0.817
Support Vector Machine	0.925	0.929	0.920	0.925	0.872	0.877	0.866	0.871
Logistic Regression	0.895	0.901	0.888	0.895	0.876	0.884	0.866	0.875
Multinomial Naïve Bayes	0.873	0.885	0.858	0.871	0.850	0.867	0.827	0.847

Further Test 1:

The table below lists the performance results of the models when we include review headings alongside the review body column for vectorization:

Model	TRAINING SET (Approximated decimals)				TEST SET (Approximated decimals)			
Name	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
Perceptron	0.946	0.944	0.949	0.946	0.895	0.894	0.896	0.895
Support Vector Machine	0.960	0.961	0.960	0.960	0.926	0.926	0.924	0.925
Logistic Regression	0.937	0.940	0.934	0.937	0.929	0.932	0.925	0.928
Multinomial Naïve Bayes	0.908	0.919	0.895	0.907	0.893	0.905	0.878	0.892

It can be inferred from the above table that the models perform better when we add review header to the review body.

Further Test 2:

The table below lists the performance results of the models when we limit the feature vectors to have a minimum frequency of 2. The vectorizer generates lower number of features indicating most of our features lie within the < 2 frequency bound.

```
red_train.shape
```

```
(160000, 35746)
```

TF-IDF Feature vector size:

Model	TRAINING SET (Approximated decimals)				TEST SET (Approximated decimals)			
Name	Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
Perceptron	0.935	0.926	0.945	0936	0.898	0.891	0.907	0.899
Support Vector Machine	0.953	0.954	0.952	0.953	0.926	0.927	0.924	0.925
Logistic Regression	0.936	0.939	0.933	0.936	0.929	0.932	0.925	0.928
Multinomial Naïve Bayes	0.903	0.907	0.899	0.903	0.894	0.897	0.890	0.893

However, as we notice in the table above there isn't too big a difference with the results with test set even with reduced features, while the performance on the test set is fluctuating.

The complete Jupyter Notebook follows this report.