CSCI - 544 Homework No. 1

NAME : ASHWIN CHAFALE

USC ID : 1990624801

1. Dataset Preparation

- Downloaded the `Amazon Reviews Dataset`
- Randomly selected `20,000` reviews from each rating class and created a balanced dataset
- Test - train split ratio : 80% & 20 %.

  Note: that the train - test split step is done in TF-IDF step

2. Data Cleaning

Data Cleaning steps done for all reviews are as follows :

- Removing `Null` values
- Converting all reviews to lower case
- Removed the HTML and URLs from the reviews
- Removed non-alphabetical characters
- Removed extra spaces
- Performed contractions using contractions library

Average Length of reviews before Data Cleaning step : `130.81`
Average Length of reviews after Data Cleaning step : `126.72`
Note : the above length may vary based on the random review samples picked for each class

3. Pre-processing

- Stop word removal
- Lemmatization

Average Length of reviews before Pre-processing step : `126.72`
Average Length of reviews after Pre-processing step : `78.67`
Note : the above length may vary based on the random review samples picked for each class

Note:
In the assignment `Stop Word Removal` data pre-processing step have been eliminated. As I have noticed that after stop-words are not removed it leads to increase in average precision of all the ML models by 10%.

4. Feature Extraction

Used sklearn to extract TF-IDF features.

5. Perceptron

Please refer the jupyter notebook below for results

6. SVM

Please refer the jupyter notebook below for results

7. Logistic Regression

Please refer the jupyter notebook below for results

8. Multinomial Naive Bayes

Please refer the jupyter notebook below for results

Libraries Used

```python
import pandas as pd
import numpy as np
import nltk
import re
from bs4 import BeautifulSoup
from nltk.corpus import stopwords
import contractions
from nltk.stem import WordNetLemmatizer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.linear_model import Perceptron
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
```

Installation Required

```python
# Installation before running the notebook
! pip install bs4
! pip install contractions
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('punkt')
```

ashwin-chafale

September 8, 2022

1 CSCI-544 Homework Assignment No. 1

1.0.1 Name : Ashwin Chafale

1.0.2 USC ID : 1990624801

1.1 Sentiment Analysis on Amazon reviews dataset

[1]:
```
import pandas as pd
import numpy as np
import nltk
import re
from bs4 import BeautifulSoup
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
# Installation before running the notebook
! pip install bs4
! pip install contractions
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('punkt')
```

```
Requirement already satisfied: bs4 in
/Users/ashwin/.conda/envs/HW1/lib/python3.10/site-packages (0.0.1)
Requirement already satisfied: beautifulsoup4 in
/Users/ashwin/.conda/envs/HW1/lib/python3.10/site-packages (from bs4) (4.11.1)
Requirement already satisfied: soupsieve>1.2 in
/Users/ashwin/.conda/envs/HW1/lib/python3.10/site-packages (from
beautifulsoup4->bs4) (2.3.1)
Requirement already satisfied: contractions in
/Users/ashwin/.conda/envs/HW1/lib/python3.10/site-packages (0.1.72)
Requirement already satisfied: textsearch>=0.0.21 in
/Users/ashwin/.conda/envs/HW1/lib/python3.10/site-packages (from contractions)
(0.0.21)
Requirement already satisfied: pyahocorasick in
/Users/ashwin/.conda/envs/HW1/lib/python3.10/site-packages (from
textsearch>=0.0.21->contractions) (1.4.4)
```

```
Requirement already satisfied: anyascii in
/Users/ashwin/.conda/envs/HW1/lib/python3.10/site-packages (from
textsearch>=0.0.21->contractions) (0.3.1)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/ashwin/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /Users/ashwin/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /Users/ashwin/nltk_data…
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package punkt to /Users/ashwin/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
```

[2]:
```
True
```

1.2 Read Data

1. Amazon reviews dataset
2. Our goal is to train sentiment analysis classifiers that can predict the rating value for a given review.

[3]:
```
df = pd.read_csv("amazon_reviews_us_Jewelry_v1_00.tsv", sep='\t', header=0,␣
↪on_bad_lines='skip')
```

1.3 Keep Reviews and Ratings

[4]:
```
df = df[['review_body','star_rating']]
df.head()
```

[4]:
```
                                         review_body star_rating
0  so beautiful even tho clearly not high end …            5
1  Great product.. I got this set for my mother, …           5
2  Exactly as pictured and my daughter's friend l…           5
3  Love it. Fits great. Super comfortable and nea…           5
4  Got this as a Mother's Day gift for my Mom and…           5
```

Removing `Null` and missing values from the dataset

[5]:
```
df = df.dropna()
df = df.reset_index(drop=True)
df.shape
```

[5]:
```
(1766748, 2)
```

1.4 We select 20000 reviews randomly from each rating class.

[6]:
```
df['star_rating'] = df['star_rating'].astype(int)

sample_size  = 20000
five_star =  df.loc[ df['star_rating'] == 5].sample(sample_size)
four_star =  df.loc[ df['star_rating'] == 4].sample(sample_size)
three_star =  df.loc[ df['star_rating'] == 3].sample(sample_size)
two_star =  df.loc[ df['star_rating'] == 2].sample(sample_size)
one_star =  df.loc[ df['star_rating'] == 1].sample(sample_size)

data = pd.concat([five_star, four_star, three_star, two_star, one_star], axis=0)
```

[7]:
```
print("Average Length of reviews before Data Cleaning step = ",␣
↪data['review_body'].str.len().mean())
```

```
Average Length of reviews before Data Cleaning step =  130.81458
```

2 Data Cleaning

2.0.1 1. Converting all reviews to lower case

[8]:
```
# convert all reviews to lower case
data["pre_processed_reviews"] = data['review_body'].apply(lambda x: " ".join(x.
↪lower() for x in str(x).split()))
```

2.0.2 2. Removing the HTML and URLs from the reviews

[9]:
```
# remove HTML tags as well as URLs from reviews.
data["pre_processed_reviews"] = data["pre_processed_reviews"].apply(lambda x:␣
↪BeautifulSoup(x).get_text())
data["pre_processed_reviews"] = data["pre_processed_reviews"].apply(lambda x:␣
↪re.sub(r"http\S+", "", x))
```

2.0.3 3. Perform "Contractions" on reviews

[10]:
```
# contractions
import contractions
data["pre_processed_reviews"] = data["pre_processed_reviews"].apply(lambda x:
↪contractions.fix(x))
```

2.0.4 4. Remove the non-alpha characters

[11]:
```
# remove the non-alpha characters
data["pre_processed_reviews"] = data["pre_processed_reviews"].apply(lambda x: "␣
↪".join([re.sub("[^A-Za-z]+","", x) for x in nltk.word_tokenize(x)]))
```

2.0.5 5. Remove extra spaces among the words

[12]:
```
# remove extra spaces among the words
data['pre_processed_reviews'] = data['pre_processed_reviews'].apply(lambda x:␣
↪re.sub(' +', ' ', x))
```

[13]:
```
print("Average Length of reviews after Data Cleaning step = ",␣
↪data['pre_processed_reviews'].str.len().mean())
```

```
Average Length of reviews after Data Cleaning step =  126.72483
```

3 Pre-processing

[14]:
```
print("Average Length of reviews before Data Pre-processing step = ",␣
↪data['pre_processed_reviews'].str.len().mean())
```

```
Average Length of reviews before Data Pre-processing step =  126.72483
```

3.0.1 1. Remove stop words

Note: Just for the purpose of pre-processing I have shown the stop-words removal. However, the stop-word removed pre-processed data is not used to train the model.

Reason for not performing stop-word removing step: *I have noticed that after stop-words are not removed it leads to increase in average precision of all the ML models by 10%.*

[15]:
```
data_copy = data.copy(deep=True)
```

[16]:
```
# remove stop words using a NLTK package
from nltk.corpus import stopwords
sw_nltk = stopwords.words('english')
sw_nltk.remove("not")
sw_nltk.remove("don")
sw_nltk.remove("don't")
sw_nltk.remove("aren't")
sw_nltk.remove("couldn't")
sw_nltk.remove("couldn")
sw_nltk.remove("didn")
sw_nltk.remove("didn't")
sw_nltk.remove("doesn")
sw_nltk.remove("doesn't")
sw_nltk.remove("won")
sw_nltk.remove("won't")
data_copy['pre_processed_reviews'] = data_copy['pre_processed_reviews'].
↪apply(lambda x: " ".join([x for x in x.split() if x not in sw_nltk]))
```

[17]:
```
print("Average Length of reviews after Data Pre-processing step = ",␣
↪data_copy['pre_processed_reviews'].str.len().mean())
```

```
Average Length of reviews after Data Pre-processing step =  78.67438
```

3.0.2 2. Perform Lemmatization

[18]:
```python
# lemmatization using wordnet lemmatizer
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
data['pre_processed_reviews'] = data['pre_processed_reviews'].apply(lambda x: "␣
↪".join([lemmatizer.lemmatize(w) for w in nltk.word_tokenize(x)]))
```

4 TF-IDF Feature Extraction

[19]:
```python
# Train - test split
from sklearn.model_selection import train_test_split

five_star_X_train, five_star_X_test, five_star_Y_train, five_star_Y_test = \
train_test_split(data[data["star_rating"] == 5]["pre_processed_reviews"],
                 data[data["star_rating"] == 5]["star_rating"], test_size=0.2,␣
↪random_state=30)

four_star_X_train, four_star_X_test, four_star_Y_train, four_star_Y_test = \
train_test_split(data[data["star_rating"] == 4]["pre_processed_reviews"],
                 data[data["star_rating"] == 4]["star_rating"], test_size=0.2,␣
↪random_state=30)

three_star_X_train, three_star_X_test, three_star_Y_train, three_star_Y_test = \
train_test_split(data[data["star_rating"] == 3]["pre_processed_reviews"],
                 data[data["star_rating"] == 3]["star_rating"], test_size=0.2,␣
↪random_state=30)

two_star_X_train, two_star_X_test, two_star_Y_train, two_star_Y_test = \
train_test_split(data[data["star_rating"] == 2]["pre_processed_reviews"],
                 data[data["star_rating"] == 2]["star_rating"], test_size=0.2,␣
↪random_state=30)

one_star_X_train, one_star_X_test, one_star_Y_train, one_star_Y_test = \
train_test_split(data[data["star_rating"] == 1]["pre_processed_reviews"],
                 data[data["star_rating"] == 1]["star_rating"], test_size=0.2,␣
↪random_state=30)

X_train = pd.concat([five_star_X_train, four_star_X_train, three_star_X_train,␣
↪two_star_X_train, one_star_X_train])
X_test = pd.concat([five_star_X_test, four_star_X_test, three_star_X_test,␣
↪two_star_X_test, one_star_X_test])
Y_train = pd.concat([five_star_Y_train, four_star_Y_train, three_star_Y_train,␣
↪two_star_Y_train, one_star_Y_train])
```

```
Y_test = pd.concat([five_star_Y_test, four_star_Y_test, three_star_Y_test,␣
    ↪two_star_Y_test, one_star_Y_test])

print("Train: ", X_train.shape, Y_train.shape, "Test: ", (X_test.shape, Y_test.
    ↪shape))
```

```
Train:  (80000,) (80000,) Test:  ((20000,), (20000,))
```

[20]:
```
# TF-IDF step
from sklearn.feature_extraction.text import TfidfVectorizer
tf_idf_vector =  TfidfVectorizer()
tf_x_train = tf_idf_vector.fit_transform(X_train)
tf_x_test = tf_idf_vector.transform(X_test)
```

5 Perceptron

[21]:
```
from sklearn.linear_model import Perceptron
from sklearn.metrics import classification_report
perceptron = Perceptron(max_iter=1000, random_state=0)
perceptron.fit(tf_x_train,Y_train)
y_test_predicted = perceptron.predict(tf_x_test)

report = classification_report(Y_test, y_test_predicted, output_dict=True)
pd.DataFrame.from_dict(report)[["1", "2", "3", "4", "5", "weighted avg"]][:3].
    ↪transpose()
```

[21]:

	precision	recall	f1-score
1	0.529361	0.46200	0.493392
2	0.302142	0.46900	0.367519
3	0.320768	0.22125	0.261873
4	0.383514	0.36175	0.372314
5	0.590234	0.55600	0.572606
weighted avg	0.425204	0.41400	0.413541

6 SVM

[22]:
```
from sklearn.svm import LinearSVC
svm = LinearSVC(multi_class="ovr", random_state=0)
svm.fit(tf_x_train,Y_train)
y_test_predicted = svm.predict(tf_x_test)

report = classification_report(Y_test, y_test_predicted, output_dict=True)
pd.DataFrame.from_dict(report)[["1", "2", "3", "4", "5", "weighted avg"]][:3].
    ↪transpose()
```

[22]:

	precision	recall	f1-score
1	0.563424	0.67625	0.614703
2	0.404890	0.33950	0.369323
3	0.424075	0.38400	0.403044
4	0.472624	0.41650	0.442791
5	0.639847	0.75150	0.691193
weighted avg	0.500972	0.51355	0.504211

7 Logistic Regression

1. Simple Logistic Regression

[23]:

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(max_iter=1000, solver='saga')
lr.fit(tf_x_train,Y_train)
y_test_predicted = lr.predict(tf_x_test)

report = classification_report(Y_test, y_test_predicted, output_dict=True)
pd.DataFrame.from_dict(report)[["1", "2", "3", "4", "5", "weighted avg"]][:3].
 ↪transpose()
```

[23]:

	precision	recall	f1-score
1	0.601127	0.66650	0.632128
2	0.427997	0.39900	0.412990
3	0.442865	0.42825	0.435435
4	0.495403	0.44450	0.468573
5	0.665905	0.72900	0.696026
weighted avg	0.526659	0.53345	0.529030

7.0.1 2. Hyper-parameter tuning for Logistic Regression

[24]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold

# define models and parameters
model = LogisticRegression(max_iter=10000)

solvers = ['newton-cg', 'lbfgs', 'liblinear']
penalty = ['l2']
c_values = [100, 10, 1.0, 0.1, 0.01]

# define grid search
grid = dict(solver=solvers, penalty=penalty, C=c_values)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv,␣
 ↪scoring='accuracy', error_score=0)
grid_result = grid_search.fit(tf_x_train, Y_train)
```

```
# summarize results
print("Best Tuning parameters : " , grid_result.best_params_)
```

```
Best Tuning parameters : {'C': 1.0, 'penalty': 'l2', 'solver': 'lbfgs'}
```

[25]:
```
grid = dict(solver=["lbfgs"], penalty=["l2"], C=[1.0])
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv,␣
 ↪scoring='accuracy', error_score=0)
grid_result = grid_search.fit(tf_x_train, Y_train)
y_test_pred = grid_search.predict(tf_x_test)

report = classification_report(Y_test, y_test_pred, output_dict=True)
pd.DataFrame.from_dict(report)[["1", "2", "3", "4", "5", "weighted avg"]][:3].
 ↪transpose()
```

[25]:

	precision	recall	f1-score
1	0.601263	0.66650	0.632203
2	0.428150	0.39925	0.413195
3	0.442865	0.42825	0.435435
4	0.495403	0.44450	0.468573
5	0.665905	0.72900	0.696026
weighted avg	0.526717	0.53350	0.529086

8 Naive Bayes

8.0.1 1. Multinomial Naive Bayes

[26]:
```
from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB()
nb.fit(tf_x_train,Y_train)
y_test_predicted = nb.predict(tf_x_test)

report = classification_report(Y_test, y_test_predicted, output_dict=True)
pd.DataFrame.from_dict(report)[["1", "2", "3", "4", "5", "weighted avg"]][:3].
 ↪transpose()
```

[26]:

	precision	recall	f1-score
1	0.592847	0.60500	0.598862
2	0.384177	0.38725	0.385707
3	0.411308	0.42375	0.417436
4	0.458140	0.42000	0.438242
5	0.657394	0.67350	0.665349
weighted avg	0.500773	0.50190	0.501119

8.0.2 2. Hyper-parameter tuning for MultinomialNB

[27]:
```
# Hyper-parameter tuning for MultinomialNB
cv_method = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=999)
grid_params = {
    'alpha': np.linspace(0.5, 1.5, 6),
    'fit_prior': [True, False]
}

mul_nom_NB = GridSearchCV(estimator=MultinomialNB(),
                          param_grid=grid_params,
                          cv=cv_method,
                          verbose=1,
                          scoring='accuracy')
mul_nom_NB.fit(tf_x_train, Y_train)
print("Best Tuning parameters : ", mul_nom_NB.best_params_)
```

```
Fitting 15 folds for each of 12 candidates, totalling 180 fits
Best Tuning parameters :  {'alpha': 1.3, 'fit_prior': True}
```

[28]:
```
grid_params = { 'alpha': [1.5], 'fit_prior': [True] }

mul_nom_NB = GridSearchCV(estimator=MultinomialNB(),
                          param_grid=grid_params,
                          cv=cv_method,
                          verbose=1,
                          scoring='accuracy')
mul_nom_NB.fit(tf_x_train, Y_train)
y_test_predicted = mul_nom_NB.predict(tf_x_test)

report = classification_report(Y_test, y_test_predicted, output_dict=True)
pd.DataFrame.from_dict(report)[["1", "2", "3", "4", "5", "weighted avg"]][:3].
↪transpose()
```

```
Fitting 15 folds for each of 1 candidates, totalling 15 fits
```

[28]:

	precision	recall	f1-score
1	0.599248	0.59775	0.598498
2	0.386182	0.39825	0.392123
3	0.412117	0.43025	0.420988
4	0.460335	0.42650	0.442772
5	0.661254	0.66175	0.661502
weighted avg	0.503827	0.50290	0.503177

[28]: