NLP_lab-7_205229118_Mahalakshmi.S

April 27, 2021

0.0.1 Lab7. Sentiment Analysis on Movie Reviews

In this lab, you will build Multinomial Naïve Bayes model for movie reviews from Rotton Tomotto Dataset.

0.1 EXERCISE-1

0.1.1 1. Open the file, 'rotten_tomato_train.tsv' and read into a DataFrame

[75]:
```
import pandas as pd
```

[76]:
```
rotten_tomato_train = pd.read_csv('rotten_tomato_train.tsv', sep='\t')
```

0.1.2 2. Print the basic statistics such as head, shape, describe, and columns

[77]:
```
rotten_tomato_train.head()
```

[77]:
```
   PhraseId  SentenceId                                             Phrase  \
0         1           1  A series of escapades demonstrating the adage …
1         2           1  A series of escapades demonstrating the adage …
2         3           1                                           A series
3         4           1                                                  A
4         5           1                                             series

   Sentiment
0          1
1          2
2          2
3          2
4          2
```

[78]:
```
rotten_tomato_train.shape
```

[78]:
```
(156060, 4)
```

[79]:
```
rotten_tomato_train.describe
```

[79]:
```
<bound method NDFrame.describe of         PhraseId  SentenceId  \
0              1           1
```

```
1              2            1
2              3            1
3              4            1
4              5            1
…           …          …
156055    156056        8544
156056    156057        8544
156057    156058        8544
156058    156059        8544
156059    156060        8544

                                                    Phrase  Sentiment
0       A series of escapades demonstrating the adage …          1
1       A series of escapades demonstrating the adage …          2
2                                                 A series          2
3                                                        A          2
4                                                   series          2
…                                                      …        …
156055                                           Hearst 's          2
156056                            forced avuncular chortles          1
156057                                   avuncular chortles          3
156058                                            avuncular          2
156059                                             chortles          2

[156060 rows x 4 columns]>
```

[80]:
```python
rotten_tomato_train.columns
```

[80]:
```
Index(['PhraseId', 'SentenceId', 'Phrase', 'Sentiment'], dtype='object')
```

0.1.3 3. How many reviews exist for each sentiment?

[81]:
```python
review=rotten_tomato_train.groupby('Sentiment').count()
review.Phrase
```

[81]:
```
Sentiment
0     7072
1    27273
2    79582
3    32927
4     9206
Name: Phrase, dtype: int64
```

0.2 EXERCISE-2

0.2.1 1. Extract 200 reviews for each sentiment, store them into a new dataframe and create a smaller dataset. Save this dataframe in a new file, say, "small_rotten_train.csv".

[82]:
```
a=rotten_tomato_train.loc[rotten_tomato_train.Sentiment == 0]
b=rotten_tomato_train.loc[rotten_tomato_train.Sentiment == 1]
c=rotten_tomato_train.loc[rotten_tomato_train.Sentiment == 2]
d=rotten_tomato_train.loc[rotten_tomato_train.Sentiment == 3]
e=rotten_tomato_train.loc[rotten_tomato_train.Sentiment == 4]
```

[83]:
```
small_rotten_train=pd.concat([a[:200],b[:200],c[:200],d[:200],e[:200]])
```

0.3 EXERCISE-3

0.3.1 1. Open the file, "small_rotten_train.csv".

[84]:
```
small_rotten_train
```

[84]:
```
      PhraseId  SentenceId                                             Phrase  \
101        102           3    would have a hard time sitting through this one
103        104           3          have a hard time sitting through this one
157        158           5  Aggressive self-glorification and a manipulati…
159        160           5    self-glorification and a manipulative whitewash
201        202           7             Trouble Every Day is a plodding mess .
…          …           …                                                  …
3744      3745         142                                  amazing slapstick
3745      3746         142                                            amazing
3847      3848         147  When cowering and begging at the feet a scruff…
3866      3867         147  gives her best performance since Abel Ferrara …
3993      3994         151  Spielberg 's realization of a near-future Amer…

      Sentiment
101           0
103           0
157           0
159           0
201           0
…           …
3744          4
3745          4
3847          4
3866          4
3993          4

[1000 rows x 4 columns]
```

0.3.2 2. The review text are stored in "Phrase" column. Extract that into a separate DataFrame, say "X".

[85]:
```
X = small_rotten_train.Phrase
```

0.3.3 3. The "sentiment" column is your target, say "y".

[86]:
```
y = small_rotten_train.Sentiment
```

0.3.4 4. Perform pre-processing: convert into lower case, remove stop words and lemmatize. The following function will help.

[87]:
```
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Mahalakshmi\AppData\Roaming\nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[88]:
```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
```

[89]:
```
def clean_review(review):
    tokens = review.lower().split()
    filtered_tokens = [lemmatizer.lemmatize(w) for w in tokens if w not in␣
 ↪stop_words]
    return " ".join(filtered_tokens)
```

0.3.5 5. Apply the above function to X

[90]:
```
temp=X.tolist()
fax=[]
```

[91]:
```
for i in temp:
    fax.append(clean_review(i))
n_X=pd.Series(fax)
```

0.3.6 6. Split X and y for training and testing (Use 20% for testing)

[92]:
```
from sklearn.model_selection import train_test_split
```

[93]:
```
X_train,X_test,y_train,y_test = train_test_split(n_X,y,train_size=0.
 ↪8,test_size=0.2)
```

0.3.7 7. Create TfidfVectorizer as below and perfrom vectorization on X_train using fit_perform() method.

[94]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

[95]:
```
tf=TfidfVectorizer(min_df=3, max_features=None,ngram_range=(1, 2), use_idf=1)
tf
```

[95]:
```
TfidfVectorizer(min_df=3, ngram_range=(1, 2), use_idf=1)
```

[96]:
```
tfi=tf.fit_transform(X_train)
tfi.shape
```

[96]:
```
(800, 904)
```

0.3.8 8. Create MultinomialNB model and perform training using X_train_lemmartized and y_train.

[97]:
```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
```

[98]:
```
X_train_mnb = cv.fit_transform(X_train)
X_test_mnb = cv.transform(X_test)
```

[99]:
```
from sklearn.naive_bayes import MultinomialNB
```

[100]:
```
clf = MultinomialNB()
```

[101]:
```
clf.fit(X_train_mnb,y_train)
```

[101]:
```
MultinomialNB()
```

0.3.9 9. Perform validation on X_test lemmatized and predict output

[102]:
```
y_lem_pred = clf.predict(X_test_mnb)
y_lem_pred
```

[102]:
```
array([2, 0, 4, 0, 3, 4, 1, 4, 3, 3, 4, 0, 0, 1, 4, 4, 1, 1, 2, 3, 3, 4,
       1, 3, 0, 2, 4, 3, 1, 3, 0, 1, 2, 3, 3, 4, 3, 3, 1, 1, 0, 2, 3, 3,
       1, 2, 1, 4, 3, 1, 4, 1, 3, 4, 1, 2, 3, 2, 2, 4, 4, 0, 3, 4, 2, 3,
       1, 2, 1, 4, 0, 1, 1, 1, 0, 0, 3, 0, 4, 0, 1, 3, 2, 1, 4, 3, 3, 0,
       2, 4, 0, 3, 3, 2, 3, 1, 4, 1, 2, 3, 2, 0, 4, 1, 1, 3, 3, 1, 1, 4,
       0, 2, 0, 2, 2, 0, 2, 0, 0, 2, 3, 3, 1, 0, 2, 0, 3, 1, 1, 1, 3, 3,
       4, 3, 4, 1, 2, 2, 1, 2, 3, 0, 1, 0, 0, 2, 0, 3, 1, 3, 4, 2, 2, 2,
       1, 4, 1, 4, 2, 4, 3, 2, 0, 3, 3, 2, 3, 1, 2, 1, 3, 0, 4, 2, 0, 1,
       4, 1, 2, 0, 2, 2, 4, 1, 3, 3, 1, 1, 3, 2, 4, 3, 3, 0, 3, 1, 1, 1,
       3, 3], dtype=int64)
```

0.3.10 10. Print classification_report and accuracy score.

[103]:
```
from sklearn.metrics import classification_report
```

[104]:
```
print(classification_report(y_test,y_lem_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.68      0.77        41
           1       0.62      0.66      0.64        44
           2       0.58      0.52      0.55        42
           3       0.47      0.69      0.56        35
           4       0.81      0.68      0.74        38

    accuracy                           0.65       200
   macro avg       0.67      0.65      0.65       200
weighted avg       0.67      0.65      0.65       200
```

[105]:
```
from sklearn.metrics import accuracy_score
```

[106]:
```
accuracy_score(y_test,y_lem_pred)
```

[106]:
```
0.645
```

0.3.11 EXERCISE-4

0.3.12 1. Open, 'rotten_tomato_test.tsv' file into dataframe

[107]:
```
rotten_tomato_test = pd.read_csv('rotten_tomato_test.tsv', sep='\t')
```

[108]:
```
rotten_tomato_test.head()
```

[108]:

	PhraseId	SentenceId	Phrase
0	156061	8545	An intermittently pleasing but mostly routine …
1	156062	8545	An intermittently pleasing but mostly routine …
2	156063	8545	An
3	156064	8545	intermittently pleasing but mostly routine effort
4	156065	8545	intermittently pleasing but mostly routine

[109]:
```
rotten_tomato_test.shape
```

[109]:
```
(66292, 3)
```

0.3.13 2. Clean this test data, using the function clean_review(), as before.

[110]:
```
X_c = rotten_tomato_test.Phrase
```

[111]:
```
t_temp=X_c.tolist()
t_fax=[]
for i in t_temp:
    t_fax.append(clean_review(i))
cr_X=pd.Series(t_fax)
```

[112]:
```
cr_X
```

[112]:
```
0         intermittently pleasing mostly routine effort .
1           intermittently pleasing mostly routine effort
2
3           intermittently pleasing mostly routine effort
4                  intermittently pleasing mostly routine
                              …
66287                long-winded , predictable scenario .
66288                  long-winded , predictable scenario
66289                                       long-winded ,
66290                                         long-winded
66291                                predictable scenario
Length: 66292, dtype: object
```

0.3.14 3. Build TFIDF values using transform() method.

[113]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

[114]:
```
tf_1=TfidfVectorizer(use_idf=True,ngram_range=(1,3),min_df = 1)
tf_1
```

[114]:
```
TfidfVectorizer(ngram_range=(1, 3))
```

[116]:
```
vec=tf_1.transform(cr_X)
vec
```

[116]:
```
<66292x904 sparse matrix of type '<class 'numpy.float64'>'
        with 68274 stored elements in Compressed Sparse Row format>
```