Project Report - Sentiment Analysis using CountVectorizer LR and TF-IDF

Introduction

We have a dataset of text data,with around more than 350,000 reviews from different customers giving their own ratings and reviews towards their own product(s) they have bought. We also do have the meta data of several such products, starting by the name of the product to its description.

We use this text-dataset to make Sentiment Analysis by using Classification Algorithms like Logistic Regression and TF-IDF.

Methods

We are going to work on this dataset using the Logistic Regression and TF-IDF Technique along with the Logistic Regression for its fit.

First of all, we begin with the required imports for analysis. Then convert our data from the dataset to a pandas data frame.

Next, we start to understand what is required from our data, and manipulate or remove values which are not useful to our goal of analysis.

Next for our general logistic regression analysis, we begin by lowercasing our 'reviewText'

```
def lowerCase(x):
    if isinstance(x, str):
        return x.lower()
    else:
        return(str(x).lower())

#df['reviewText'].apply(lambda x: lowerCase(x))
#df['summary'].apply(lambda x: lower)
df.dropna(subset=['reviewText', 'summary'])
```

Then we also remove any newline strings, which are not useful in our text.

```
def remove_newline(x):
    if isinstance(x, str):
        return x.replace('\n','')
    else:
        return(str(x).replace('\n',''))
```

Next, we remove the punctuations,

```
import string
def remove_punctuations(text):
    for punctuation in string.punctuation:
        text = text.replace(punctuation, '')
    return text
```

And then we do the stopword removal from the NLTK library.

```
import nltk
nltk.download("stopwords")
from nltk.corpus import stopwords
stop = stopwords.words('english')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/matcha.s/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
stop = [remove_punctuations(word) for word in stop] #
```

```
df['reviewText']=df['reviewText'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop)]))
df['summary']=df['summary'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop)]))
```

We can then either go with stemming or lemmatization. I went with stemming the text corpus.

```
from nltk.stem import PorterStemmer
from nltk.stem.snowball import SnowballStemmer
ps = PorterStemmer()
sns= SnowballStemmer("english")

df['reviewText']=df['reviewText'].apply(lambda x: " ".join([sns.stem(w) for w in x.split()]))
df['summary']=df['summary'].apply(lambda x: " ".join([sns.stem(w) for w in x.split()]))
```

Then I made the classification of the response variable using a custom method.

```
def applyCategory(x):
    x=int(x)
    if(x==5 or x==4):
        return "Positive"
    elif(x==3):
        return "Neutral"
    else:
        return "Negative"

df["overall"] = df["overall"].apply(lambda x: applyCategory(x))
```

Logistic Regression

For implementation, I made the train-test split of the dataset using scikit-learn. (66-33 = Train-test-split)

```
from sklearn.model_selection import train_test_split
X1 = df['reviewText']
X2 = df['summary']
y = df['overall']
X_train, X_test, y_train, y_test = train_test_split(X1, y, test_size=0.33,random_state=12)
print('Training Data :', X_train.shape)

print('Testing Data : ', X_test.shape)
```

I utilized the count vectorizer from the same library, to make the fit.

```
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import CountVectorizer
lr = LogisticRegression(max_iter=1000)

cv = CountVectorizer()

X_train_converted = cv.fit_transform(X_train)

print(X_train_converted.shape)

lr.fit(X_train_converted, y_train)
```

TF-IDF

After making the split (which was done before), I have used the TfIdfVectorizer from Scikit-learn to produce my fit

```
from sklearn.feature_extraction.text import TfidfVectorizer
v = TfidfVectorizer(max_features=1000,smooth_idf=True)
x_rt = v.fit_transform(df['reviewText'])
x_summary=v.fit_transform(df['summary'])
x_rt_array = x_rt.todense()
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import CountVectorizer
lr = LogisticRegression(max_iter=1000)
X_train_new=X_train_new.values.reshape(-1,1)
lr.fit(X_train_new, y_train_new)
```

Results

Logistic Regression

```
from sklearn.metrics import classification_report

report = classification_report(y_test, predictions)
print(report)
```

```
              precision    recall  f1-score   support

    Negative       0.72      0.61      0.66     19637
     Neutral       0.40      0.17      0.24      9687
    Positive       0.87      0.96      0.91     93220

    accuracy                           0.84    122544
   macro avg       0.67      0.58      0.60    122544
weighted avg       0.81      0.84      0.82    122544
```

Accuracy using logistic regression has turned out be 0.84.

TF-IDF

```
              precision    recall  f1-score   support

    Negative       0.00      0.00      0.00     19637
     Neutral       0.00      0.00      0.00      9687
    Positive       0.76      1.00      0.86     93220

    accuracy                           0.76    122544
   macro avg       0.25      0.33      0.29    122544
weighted avg       0.58      0.76      0.66    122544
```

Accuracy using TF-IDF has turned out to be 0.76.

Conclusion

In logistic regression, the accuracy is high enough on the overall dataset, but the precision (especially for the neutral category) can be improved by balancing on the data size by categories (Negative, Positive and Neutral)

The TF-IDF Scores are improper due to the length of the data we have analyzed. The best way to counter something like that is to smoothen the fit by preventing ‘zero-division’.

We have observed that simple logistic regression using count vectorizer can work enough over TF-IDF for our dataset.

References (cite all the sources)

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
https://nijianmo.github.io/amazon/index.html

Justifying recommendations using distantly-labeled reviews and fined-grained aspects
Jianmo Ni, Jiacheng Li, Julian McAuley
Empirical Methods in Natural Language Processing (EMNLP), 2019