PROJECT REPORT

ON

RATING PREDICTION

Submitted by: NIMESH KUMAR ROY

INTRODUCTION

The rise in E — commerce, has brought a significant rise in the importance of customer reviews. There are hundreds of review sites online and massive amounts of reviews for every product. Customers have changed their way of shopping and according to a recent survey, 70 percent of customers say that they use rating filters to filter out low rated items in their searches.

The ability to successfully decide whether a review will be helpful to other customers and thus give the product more exposure is vital to companies that support these reviews, companies like Google, Amazon,Flipkart etc.

The approach is based on review text content analysis and uses the principles of natural language process (the NLP method). This method lacks the insights that can be drawn from the relationship between costumers and items.

The data

The dataset used scraped from amazon.in and flipkart.com. It contains product name, reviews, ratings and profile name , including 30000 rows of data.

The product field contains the name of the product reviewed in this case it is oneplus and redmi.

The review field contain reviews from different users.

The rating field contain ratings given by the users according to their experience.

The Profile name contain the name of user who reviewed the product and gives the rating.

The rating field contains rating from 1-5 in it.

Data Preprocessing

Preprocessing is the key step in any data analysis. It include several steps like Noise Removal,Tokenizitaion,Normalization,StopWord Removing,Stemming,Lemmatization.

Noise Removal

Text cleaning is a technique that developers use in a variety of domains. Depending on the goal of your project and where you get your data from, you may want to remove unwanted information, such as:

- Punctuation and accents
- Special characters
- Numeric digits
- Leading, ending, and vertical whitespace
- HTML formatting

Tokenization

For many natural language processing tasks, we need access to each word in a string. To access each word, we first have to break the text into smaller components. The method for breaking text into smaller components is called tokenization and the individual components are called tokens.

Normalization

Tokenization and noise removal are staples of almost all text pre-processing pipelines. However, some data

may require further processing through text normalization. Text normalization is a catch-all term for various text pre-processing tasks. In the next few exercises, we'll cover a few of them:

- Upper or lowercasing
- Stopword removal
- Stemming – bluntly removing prefixes and suffixes from a word

Stopword Removal

Stopwords are words that we remove during preprocessing when we don't care about sentence structure. They are usually the most common words in a language and don't provide any information about the tone of a statement. They include words such as "a", "an", and "the".

NLTK provides a built-in library with these words.

Stemming

In natural language processing, stemming is the text preprocessing normalization task concerned with bluntly removing word affixes (prefixes and suffixes). For example, stemming would cast the word "going" to

“go”. This is a common method used by search engines to improve matching between user input and website hits.

NLTK has a built-in stemmer called PorterStemmer. You can use it with a list comprehension to stem each word in a tokenized list of words.

Lemmatization

Lemmatization is a method for casting words to their root forms. This is a more involved process than stemming, because it requires the method to know the part of speech for each word. Since lemmatization requires the part of speech, it is a less efficient approach than stemming.

In the next exercise, we will consider how to tag each word with a part of speech. In our case we use a package named

“preprocess.kgptalkie” .We define a

Function name “get_clean” which include all the data preprocessing steps in it.

```
import preprocess_kgptalkie as ps
import re
```

```
def get_clean(x):
    x = str(x).lower().replace('\\', '').replace('
    x = ps.cont_exp(x)
    x = ps.remove_emails(x)
    x = ps.remove_urls(x)
    x = ps.remove_html_tags(x)
    x = ps.remove_rt(x)
    x = ps.remove_accented_chars(x)
    x = ps.remove_special_chars(x)
    x = re.sub("(.)\\1{2,}", "\\1", x)
    return x
```

MODEL SELECTION AND CLASSIFICATION REPORT

We have used 7 machine learning algorithms

1) Linear SVC.
2) LogisticRegression.
3) KNeighbors Classifier.
4) Decision Tree Classifier.
5) BernoulliNB.
6) MultinomialNB.
7) RandomForest Classifier.

Every model's classification report is given below-

LINEAR SVC -

	precision	recall	f1-score	support
1	0.44	0.46	0.45	436
2	0.23	0.27	0.25	385
3	0.31	0.28	0.29	831
4	0.38	0.39	0.38	1721
5	0.66	0.66	0.66	2750
accuracy			0.49	6123
macro avg	0.41	0.41	0.41	6123
weighted avg	0.49	0.49	0.49	6123

LOGISTIC REGRESSION -

```
1 lr_pred=lr.predict(X_test)
```

```
1 print(classification_report(y_test,lr_pred))
```

	precision	recall	f1-score	support
1	0.54	0.48	0.51	436
2	0.40	0.08	0.13	385
3	0.36	0.33	0.35	831
4	0.40	0.28	0.33	1721
5	0.62	0.84	0.71	2750
accuracy			0.54	6123
macro avg	0.46	0.40	0.40	6123
weighted avg	0.50	0.54	0.50	6123

KNEIGHBORS CLASSIFIER-

```
1 print(classification_report(y_test,knn_pred))
```

	precision	recall	f1-score	support
1	0.38	0.41	0.40	436
2	0.20	0.13	0.15	385
3	0.28	0.27	0.27	831
4	0.35	0.37	0.36	1721
5	0.63	0.64	0.64	2750
accuracy			0.47	6123
macro avg	0.37	0.36	0.36	6123
weighted avg	0.46	0.47	0.46	6123

DECISION TREE CLASSIFIER -

```
dtc=DecisionTreeClassifier(criterion='entropy')
dtc.fit(X_train,y_train)
```

```
DecisionTreeClassifier(criterion='entropy')
```

```
dtc_pred = dtc.predict(X_test)
print(confusion_matrix(y_test,dtc_pred))
print(accuracy_score(y_test,dtc_pred))
print(classification_report(y_test,dtc_pred))
```

```
[[ 175   74   79   48   60]
 [  84   84  110   56   51]
 [  89   64  225  240  213]
 [  60   43  203  583  832]
 [  44   31  138  653 1884]]
0.48195329087048832
              precision    recall  f1-score   support

           1       0.39      0.40      0.39       436
           2       0.28      0.22      0.25       385
           3       0.30      0.27      0.28       831
           4       0.37      0.34      0.35      1721
           5       0.62      0.69      0.65      2750

    accuracy                           0.48      6123
   macro avg       0.39      0.38      0.39      6123
weighted avg       0.47      0.48      0.47      6123
```

BERNAULLINB -

```
bnb = BernoulliNB()
bnb.fit(X_train, y_train)
```

```
BernoulliNB()
```

```
bnb_pred = bnb.predict(X_test)
print(confusion_matrix(y_test,bnb_pred))
print(accuracy_score(y_test,bnb_pred))
print(classification_report(y_test,bnb_pred))
```

```
[[ 199   80   75   31   51]
 [  87   77  127   28   66]
 [ 105  120  254  115  237]
 [  94  106  216  367  938]
 [ 112   64  143  351 2080]]
0.4861995753715499
              precision    recall  f1-score   support

           1       0.33      0.46      0.39       436
           2       0.17      0.20      0.19       385
           3       0.31      0.31      0.31       831
           4       0.41      0.21      0.28      1721
           5       0.62      0.76      0.68      2750

    accuracy                           0.49      6123
   macro avg       0.37      0.39      0.37      6123
weighted avg       0.47      0.49      0.47      6123
```

MULTINOMIALNB -

```
1 mnb = MultinomialNB().fit(X_train, y_train)
```

```
1 mnb_pred = mnb.predict(X_test)
```

```
1 print(classification_report(y_test,mnb_pred))
```

	precision	recall	f1-score	support
1	0.52	0.36	0.43	436
2	0.20	0.01	0.01	385
3	0.32	0.39	0.35	831
4	0.38	0.29	0.33	1721
5	0.63	0.80	0.70	2750
accuracy			0.52	6123
macro avg	0.41	0.37	0.37	6123
weighted avg	0.48	0.52	0.49	6123

From the above figures we can see that logistic regression and multinomialNB both are working little bit well among these. The result is not so good but acuuracy is above 50%. We know this is not the good figure in terms of accuracy,but we can try further more and there is always a possibilitie of improvement in the model we create.
So we select the logistic regression as our model.

The predicted result is good not best.

```
x='the product is very good,price worth it'
x=get_clean(x)
vec=tfidf.transform([x])
clf.predict(vec)
```

```
array([4])
```

```
x='the product is very bad, don,t buy it'
x=get_clean(x)
vec=tfidf.transform([x])
clf.predict(vec)
```

```
array([1])
```

```
x='charger not included'
x=get_clean(x)
vec=tfidf.transform([x])
clf.predict(vec)
```

```
array([3])
```

```
x='so good,buy it'
x=get_clean(x)
vec=tfidf.transform([x])
clf.predict(vec)
```

```
array([5])
```

```
x='the product is very good,price worth it'
x=get_clean(x)
vec=tfidf.transform([x])
lr.predict(vec)
```

```
array([5])
```

```
import pickle
filename='rating_pred.pkl'
pickle.dump(lr,open(filename,'wb'))
```

```
loaded_model=pickle.load(open('rating_pred.pkl','rb'))
result=loaded_model.score(X_test,y_test)
print(result*100)
```

```
53.829821982688216
```

```
conclusion=pd.DataFrame([loaded_model.predict(X_test)[:],y_pred[:]],index=["Predicted","ori
conclusion
```

	0	1	2	3	4	5	6	7	8	9	...	6113	6114	6115	6116	6117	6118	6119	6120	6121	6122
Predicted	5	5	5	5	4	4	5	5	3	3	...	5	5	4	5	3	5	4	5	5	5
original	5	5	5	5	2	4	5	5	3	2	...	5	4	4	5	3	2	4	5	5	4

2 rows x 6123 columns

Yes again we can improve the model , this is not the best model.