GROUP-28

==========

RECOMMENDATION SYSTEM AND SENTIMENT ANALYSIS ON AMAZON FINE FOOD REVIEW DATASET

==========

INTRODUCTION

In this project we have done “Sentiment Analysis” on review of customers on amazon products, where we analyze the sentiment of the particular review whether it is positive or negative. The ground truth whether the particular review is positive or negative depends on the rating of the product given by the user.

Another is “Recommendation System” where on the basis of rating given to the product by the users. Here we look for the mutual use of two or more users and if their products are common then we will recommend their non-common product to each other.

DATASET

The dataset is “Amazon food product review” from kaggle. This dataset consists of reviews of fine food products from amazon. Dataset includes reviews from Oct 1999 to Oct 2012. Dataset having 74258 products, 256059 users, 260 users> 50 reviews, 568454 reviews (rows overall in the dataset).
Its dataset having following columns:

```
df.columns
```

```
Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
       'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text'],
      dtype='object')
```

We are mostly focused on “ProductId”, “Score”,”Text”. Where “ProductId” is the ID of all the food products for which rating is done by the customer is present in column “Score” between range 1 to 5, along with products review is present in “Text” column where reviews of particular products are given by the user.

IDEAS IMPLEMENTED

[A] SENTIMENT ANALYSIS

i.Data Preparation:

In sentiment analysis we have to use basically two columns Text and Score.

-There are lots of spaces and NaN(Non a number values i.e. null values) values in so I filled them. Like in Text where sentences are present I filled it with spaces and in the “Score” column fill null values with 0.0 .

```
df['Text']=df['Text'].fillna("")
df['Score']=df['Score'].fillna(0.0)
```

After that I look for the Score columns where they range from 1 to 5. So I converted them into two classes 0 and 1. Where 0 depicts negative response & 1 depicts positive response.

Final data after preparation which we will going to use is below:

	Reviews	Positivity
0	bought sever vital can dog food product found ...	1
1	product arriv label jumbo salt peanut peanut a...	0
2	confect around centuri light pillowi citru gel...	1
3	look secret ingredi robitussin believ found go...	0
4	great taffi great price wide assort yummi taff...	1

ii.Data preprocessing:

As now data present on the current text column is non-preprocessed. Now we have to preprocess the “Text” column so that we can focus on only important words. Like following preprocessing were applied:

1.Remove Urls: Here we remove Url links and substitute it by null spaces.

2.Remive Numbers: Here except capital and small letters words or letters I removed all the texts because there is no need of numbers in analysing positivity or negativity of a sentence words or alphabets are a major focus.

3.Upper to Lower case conversion: It's better to keep cases of the alphabets uniform so that if two words like “YES” will be equal to “yes” etc.

4.split sentences: we have to split sentences into words that may say tokenized.

5.Porterstemmer : We applied porter stemmer so that the vocabulary words should only present there and the unnecessary append of words like “ies” in “studies” get remove and will become “stud” sp that two same words just differ by appending terms should not count twice.

6.Stopwards removal: As they are most occurring words does’t contribute in deciding sentiment they are neutral words.

iii.Vectorization:

As now we have preprocessing text data that consist of important vocabulary words only now we have to convert them in number format so for that we have used following 2 vectorization techniques:

1.Tfidf Vectorization:

The **Term frequency inverse document frequency Vectorizer** will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allow you to encode new documents.To give importance to rare words.

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer()
tfidf.fit(result['Reviews'])
```

Here tf idf value jumbo is greater than peanuts mens jumbo is rare then peanuts .

```
print([doc_vector_tfidf[1, tfidf.vocabulary_['peanut']]])
```

```
[0.33186936733548433]
```

```
print([doc_vector_tfidf[1, tfidf.vocabulary_['jumbo']]])
# jumbo is rare word having high tf-idfvalue:
```

```
[0.6042910499832276]
```

2.CountVectorization:

Count the terms Vectorizer is a great tool provided by the scikit-learn library in **Python**. It is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text.

```
cv = CountVectorizer()
```

IV.TEST-TRAIN split:

We split all the dataset into the test train to identify the accuracy of the train models. Split details below.

```
Overall train data  37500 entries, 23.106666666666666 negativ response and  76.89333333333333  positive response
```

```
Overall train data  12500 entries, 23.336000000000002 negativ response and  76.664  positive response
```

V.MODEL APPLIED:

1.KNN:

The K-nearest neighbor algorithm is a Supervised machine learning algorithm.It is very easy to implement and performs classification even though is complex.KNN does not assume anything about the data which in technical terms says nonparametric learning algorithm.It’s called lazy learner.It didn't have any specialised training phase.It uses the data point of whole data while training and and classifies the new data instances.It uses euclidean distance as a metric.It

uses its nearest neighbor to predict the class label. In text classification there is use of nltk libraries to generate similarities and similarity scores which will be taken out among all texts.In this model we find out the highest similarity score which among all the training data sets.Here It will learn from its neighbour, say instance based model. Here I used k neighbor as 7.

2.Multinomial Naive Bayes classifier:

It is said to be the baseline solution for sentimental analysis tasks.Naive bayes technique is to find probabilities of classes which are assigned to document texts by using joint probabilities of all the labeled classes. Training multinomial classifier follow a sequence of technique as: vectorizer to transformer to Classifier is easy to work.It is using word frequency as a feature so on the basis of how many time that word will appear on the whole document it will classifies the text in classes.The following is the basic probability formula for naive bayes classifier-

P(label|features)=P(label)*P(features|label)/P(features)

3.Random forest:

Bagging taking random samples of data from the dataset and train model over that.We have used here random model to bagged the result. Random forest takes a lot of different trees and according to their majority prediction it will predict the classes. Result with low bias and low variance.

4.Major voting classifier:

Above 3 models are applied here and with less bias and variance on the basis of majority classification of prediction it will give output.

VI.Evaluation Metric :

Tp:true positive , Tn=true negative, fp=false positive, fn=false negative.

1.Accurcay
2.Precision(pre)
3.Recall
4.F1-Score

Formulas for above 4 metric is:

Accuracy=(tp+fn)/(tp+tn+fp+fn)

```
pre=tp/(tp+fp)
recall=tp/(tp+fn)
f1_score=(2*pre*recall)/(pre+recall)
```

VII.RESULT:

Multinomial naive Bayes gives us highest accuracy among all models in countvectorization but Recall will be highest in Voting classification that is 99% which is more preferable here. Voting classification can be considered the best model. Recall is the intuitive ability to classify to find all positive samples.

	Voting classifier	MultinomialNB	KNN	Random forest
Tf-idf:				
Accuracy:	83.02	83.03	79.8	85.9
Precision:	82.15	82.5	81.8	85
Recall:	99.4	98.7	94.2	98
F1-Score:	89.9	89.9	87.6	91
Countvectorization:				
Accuracy:	82.99	85.55	76.2	84.824
Precision:	81	82.5	81.8	85
Recall:	99	98.7	94.23	98
F1-Score:	89.6	89.99	87.2	91.39

VIII. User Interface:

Natural Language interface for users to write a statement and check whether the commented review is positive or negative.Following is how it looks like while running the code.

Supposed user wrote some review “The food is delicious everything color, taste , flavour is awesome”, then our model will return in response “Positive response” else negative response this is what sentiment is about.

USER INTERFACE

Here user have to give input a review, and then our models will predict whether the review given to the product is positive or negative response:


```
Enter the review to check whether it is positive or negative:
The food is very delicious color flavour and taste everything is awsome.
PRECISION,RECALL and F1-SCORE:  0.821582485778314 , 0.9946780757591568 , 0.8998819919754543
accuracy score:  83.032
Positive Response
```

[B] SENTIMENT ANALYSIS(FEATURE ENGINEERING)

Here we extract some features from review texts to predict it's sentiment instead of considering it as a simple document vector in terms of TF-IDF or Vocab Count.

i.Data Preparation:

In sentiment analysis we have to use basically two columns Text and Score.
So we make a data frame that has only two columns 'Text ' and 'score'.

For Feature engineering implementation of sentiment analysis, first 150000 rows are taken as training dataset and next 150000 rows are taken as test dataset. There are lots of spaces and NaN(Non a number values i.e. null values) values in the dataset. Like in Text (where sentences are present)if there is any NAN value,I fill it with spaces and in the "Score" column(contains float value) fill null values with 0.0 .

```
df['Text']=df['Text'].fillna("")
df['Score']=df['Score'].fillna(0.0)
```

The review who has 'Score' greater than 3 is considered as positive sentiment, and has 'Score' less than 3, is considered as negative sentiment. So we added one more column named 'Sentiment' to the dataset , if a review has scored more than 3, it is given to 1 else 0. Drop the score column from the data set.

```
r=[]
for i in df_review['Score']:
  if i>3:
    r.append(1)
  else:
    r.append(0)
```

ii.Data Preprocessing:

1. Remove extra spaces from text data.
2. Used NLTK Word tokenizer to generate tokens from text data.
3. Remove stopwords(used nltk stop word list) from token lists.
4. Use nltk word lamatizer to lamatize the tokens.

```
def preprocessing(sen):
  # result=sen.lower()
  result = re.sub(r"\d+", "", result)
  stop_words = set(stopwords.words('english'))
  tokens = nltk.word_tokenize(result)
  result = [i for i in tokens if not i in stop_words]
  lemmatizer=nltk.WordNetLemmatizer()
  doc=[]
  for word in result:
    word1=lemmatizer.lemmatize(word)
    doc.append(word1)
  return doc
```

iii.Feature Engineering:

Each review data was represented as a set of following features.

1.Word ngrams:
presence or absence of contiguous sequences of 1, 2 tokens; noncontiguous ngrams (ngrams with one token replaced by *). For noncontiguous we considered only bigram.(we used nltk ngram and skip gram library to generate contiguous and noncontiguous n gram respectively)

```
def skipgrams(ind,sen):
  global all_ngram
  global ngram
  sen1=[]
  for i in sen:
        i=re.sub('[!@#$%^&*()|\/,.:;<>-?\'\"]','',i)
        sen1.append(i)
  sen=sen1
  skip_bigram=list(nltk.skipgrams(sen,2,4))
    # skip_trigram=list(nltk.skipgrams(sen,3,4))
  temp={}
  for col in skip_bigram:
      temp[col]=1
      try:
        all_ngram[col]+=1
      except:
        all_ngram[col]=1
  ngram[ind]=temp
```

```
def unigrams(id1,sen):
  global all_unigram
  global unigram
  sen1=[]
  for i in sen:
    i=re.sub('[!@#$%^&*()|\/,.:;<>-?\'\"]','',i)
    sen1.append(i)
  sen=sen1
  unigram_=list(nltk.ngrams(sen,1))
  temp={}
  for col in unigram_:
    temp[col]=1
    try:
      all_unigram[col]+=1
    except:
      all_unigram[col]=1
  unigram[id1]=temp
```

2.POS:

We have calculated the number of occurrences of each partof-speech tag in each review.

```
def pos(sen):
    temp={}
    tag=set()
#     tokens=nltk.word_tokenize(sen)
    for w1 in sen:
        w1=re.sub('[!@#$%^&*()|\/,.:;<>-?]','',w1)
    pos_=nltk.pos_tag(sen)
    for pair in pos_:
        if pair[1] not in temp:
            temp[pair[1]]=1
    return temp
```

3.Punctuation:

The number of contiguous sequences of exclamation marks, question marks, and both exclamation and question marks present in review.

Whether the last token contains an exclamation or question mark.

```
def punctuation(sen):
    c=0

    for word in sen:
        if re.search('[!?]+',word):
            c+=1
    pl=0
    if len(sen)>0 and re.search('[!?]+',sen[-1]):
        pl=1
    return c,pl
```

4.Emoticons:

Presence or absence of positive and negative emoticons at any position in the review, whether the last token is a positive or negative emoticon.

(The polarity of an emoticon was determined with a regular expression adopted from Christopher Potts' tokenizing script)

```
def emoticons(sen):
  emoticon_string = r"""
    (?:
      [<>]?
      [:;=8]
      [\-o\*\']?
      [\)\]\(\[dDpP/\:\}\{@\|\\]
      |
      [\)\]\(\[dDpP/\:\}\{@\|\\]
      [\-o\*\']?
      [:;=8]
      [<>]?
    )"""
  emojies=re.findall(emoticon_string,sen)
  temp={}
  for em in emojies:
    if em not in temp:
      temp[em]=1
  l_em=0
  if re.search(emoticon_string,sen[-1]):
    l_em=1
  return temp,l_em
```

5.elongated words:

The number of words with one character repeated more than two times, for example, ‘Gooood’.

```
def elongation(sen):
    elong_string="([a-zA-Z])\\1{2,}"
    c=0
#     sen=sen.split(' ')
    for w in sen:
        if re.search(elong_string,w):
            c+=1
    return c
```

6.All-caps:

The number of words with all characters in upper case;

```
def allcap(sen):
#     sen=re.sub('_-',' ',sen)
#     sen=sen.split()
    cap='^[#@$\n]*[A-Z]+[.?!-_,.]*[A-Z]*$'
    c=0
    for word in sen:
        if re.search(cap,word):
            c+=1
    return c
```

8.negation:

The number of negated contexts present in review.

```
def negation(sen):
    negation_se='(?:^(?:never|no|nothing|nowhere|noone|none|not|hav
#     sen=sen.split()
    c=0
    for i in range(len(sen)-1):
        if re.search(negation_se,sen[i]):
            c+=1
    return c
```

Snapshot of above features

	JJ	VBP	RB	VBG	NN	.	VB	IN	punc_count	last_punc	last_emo	elongation
8	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0
13	1.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	3.0	1.0	0.0	0.0
14	1.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0
17	1.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
19	1.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
149928	1.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
149930	1.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0	2.0	1.0	0.0	0.0
149933	1.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0
149984	1.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
149988	1.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0

13869 rows × 47 columns

	arrived	quickly	perfect	condition		definitely
150039	1.0	1.0	1.0	1.0	1.0	1.0
150059	0.0	0.0	0.0	0.0	1.0	0.0
150061	0.0	0.0	0.0	0.0	1.0	0.0
150105	0.0	0.0	0.0	0.0	1.0	0.0
150147	0.0	0.0	0.0	0.0	1.0	0.0
...	...	...	...	...	...	...
299985	0.0	0.0	0.0	0.0	1.0	0.0
299988	0.0	0.0	0.0	0.0	1.0	0.0

8. Lexicon:

We have taken NRC Emotion Lexicon, MPQA, Bing Liu Lexicon script. These scripts contain words and it's polarity in terms of different dimensions.

So if any word from a review data present in the above mentioned script, we take the polarity of that word(for all dimensions), and in that way we have calculated how much overall polarity the review has. Suppose, "Good" and "less" these two words present in review data, and we get polarity score 1 for "Good"

and “-1” for “less”, or get “GOOd” as positive polarity and “less” as negative polarity word.

Structure of each script file:

NRC Emotion Script:

```
'baggage': {'anger': '0',
 'anticipation': '0',
 'disgust': '0',
 'fear': '0',
 'joy': '0',
 'negative': '0',
 'positive': '0',
 'sadness': '0',
 'surprise': '0',
 'trust': '0'},
```

BingLiu Script:

	Word	Polarity
0	a+	positive
1	abound	positive
2	abounds	positive
3	abundance	positive
4	abundant	positive
...	...	...

MPQA Script:

```
df_mpqa
```

	Word	Polarity
0	abandoned	negative
1	abandonment	negative
2	abandon	negative
3	abase	negative
4	abasement	negative
...	...	...

Snapshot of Lexicon feature:

	positive	negative	surprise	joy	positive	trust	negative	fear	sadness	anticipation	anger
8	2	0	0	1	2	0	0	0	0	0	0
13	5	0	1	3	3	1	0	0	0	1	0
14	1	1	0	0	0	0	2	0	1	0	2
17	2	0	0	1	1	0	0	0	0	0	0
19	3	1	0	1	2	1	2	0	1	2	0
...	...	...	...	...	...	...	...	...	...	...	...

Vi.Model Implementation:

MLP:

“A multilayer perceptron (MLP) is a feedforward artificial neural network model that maps sets of input data onto a set of appropriate outputs.
We get maximum accuracy for this model, it may happen as it has multiple layers. “

Decision Tree:

“A **decision tree** is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label”
This is a rule based classifier, so it makes rules depending on the value of the feature.

SVM:

“**SVM** is a supervised machine learning algorithm which can be **used** for classification or regression problems. It uses a technique called the kernel trick to transform data and then based on these transformations it finds an optimal boundary between the possible outputs.”

V.Evaluation Metrics:

Tp:true positive , Tn=true negative, fp=false positive, fn=false negative.
1.Accurcay
2.Precision(pre)
3.Recall

4.F1-Score
Formulas for above 4 metric is:

Accuracy=(tp+fn)/(tp+tn+fp+fn)

```
pre=tp/(tp+fp)
recall=tp/(tp+fn)
f1_score=(2*pre*recall)/(pre+recall)
```

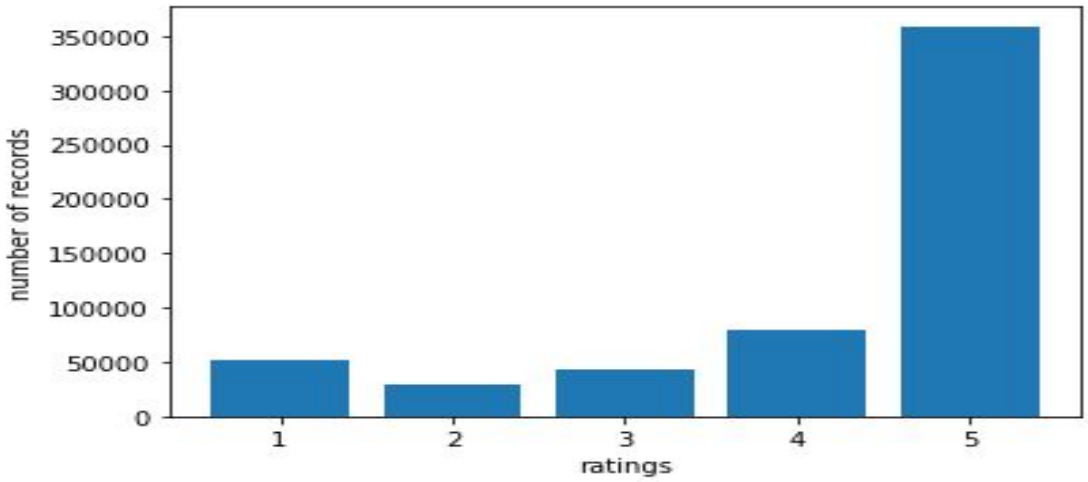
Vi.Result:

	SVM	DECISION TREE	MLP
Accuracy:	89.4	85.7	90.2
Precision:	90.1	91.6	92.9
Recall:	97.9	90.9	95.5
F1-Score:	93.9	91.3	94.2

[C].Recommendation of product based on the user ratings:

Data Preprocessing and Preparation:

For the recommendation system we generally need the reviews data, customer_id and product id. For content based recommendation we need review text while we need review score only when we do collaborative based recommendations. **Collaborative filtering** is a family of algorithms where there are multiple ways to find similar users or items and multiple ways to calculate rating based on ratings of similar users. So, we deleted the rest of columns like Id, ProfileName, HelpfulnessNumerator, HelpfulnessDenominator, Time, Summary and Text.
The count of each type of score is given in the figure below.



For better recommendation, we deleted the product which is rated by less than 30 users.
We also deleted the user who has rated who has rated less than 10 food products. So in the end we are left out with 78,737 rows.

Procedure

- For new users, we followed the most common approach in which we show the most popular product as recommendations. We counted the number of

ratings on each product and recommended the 10 most popular food products.

- For existing users, We first reshaped the dataframe to a new dataframe having user_id in the rows and each product_id in the columns, where each cell contained the rating given by user.
  Then we find the similarity between each item's rating using correlation matrices.
  Now we separated the scores of the similar product to ones which were rated by the users from the correlation matrix and showed them as recommendations for that user.
  We recommended the 10 food products to the users.

Results

- We have shown the recommendation results for both the new users as well as the old users. The id of old users are AY12DBB0U420B and A2SZLNSI5KOQJT and the new users are 0 and 1.
  The recommendation results for existing users can be seen in below figure:

```
recommendation for user AY12DBB0U420B is:
B001AHJ2D8
B001AHJ2FQ
B001AHFVHO
B000YSRK7E
B000YSTIL0
B00248EE4O
B001AHL6CI
B000N30EC8
B001BCVY4W
B001BCVY9W

recommendation for user A2SZLNSI5KOQJT is:
B002HQLY7S
B003N0ZEKU
B007TJGZ0Y
B0033HPPIY
B007OXJKF2
B007RTR8AM
B007OXJJQ2
B00370CFR6
B001RVFDOO
B007OXJL0G
```

The recommendation results for existing users can be seen in below figure:

```
recommendation for user 0 is:
B007JFMH8M
B003B3OOPA
B0026RQTGE
B002QWHJOU
B002QWP89S
B002QWP8H0
B001EO5Q64
B001RVFERK
B007M83302
B007M832YY

recommendation for user 1 is:
B007JFMH8M
B003B3OOPA
B0026RQTGE
B002QWHJOU
B002QWP89S
B002QWP8H0
B001EO5Q64
B001RVFERK
B007M83302
B007M832YY
```

==========

Work Distribution

Ritesh Singh(MT19044): Recommendations System[C]
Akanksha Dewangan(MT19049): Sentiment Analysis with User interface(UI)[A]
Anamitra Maji(MT19112): Sentiment Analysis with Feature engineering(novelty).[B]

==========

Software Used:

Anaconda, Pycharm and Jupyter.

==========

URLS:

1.https://www.kaggle.com/snap/amazon-fine-food-reviews

2.https://www.kaggle.com/saurav9786/recommendation-based-on-food-review

3.https://www.kaggle.com/aniruddhachoudhury/sentiment-classification-with-nlp-on-lstm

4.https://www.kaggle.com/kshitijmohan/sentiment-analysis-universal-sentence-encoder-91

5.https://www.kaggle.com/saurav9786/recommendation-based-on-food-review

6.https://www.google.com/search?sxsrf=ALeKk019kJZlLptGf3yka_72H7neD5Y4fw%3A1607256577725&ei=AcrMX_fgK9HJpgfwjJzQCg&q=countvectorizer&oq=countvect&gs_lcp=CgZwc3ktYWIQAxgAMgoIABCxAxDJAxBDMgQIABBDMgQIABBDMgQIABBDMgQIABBDMgQIABBDMgIIADICCAAyAggAMgIIADoECCMQJzoHCAAQyQMQQzoFCAAQsQNQ2bwDWNLPA2C22ANoAHACeACAAaACiAGWDZIBBTAuNy4ymAEAoAEBqgEHZ3dzLXdpesABAQ&sclient=psy-ab