code_ML

July 5, 2022

[80]:
```
# Libraries
import sys
import pandas as pd
import nltk
from sklearn.metrics import classification_report
import html
import numpy as np
import re
import warnings
warnings.filterwarnings('ignore')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('omw-1.4')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\wajah\AppData\Roaming\nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\wajah\AppData\Roaming\nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\wajah\AppData\Roaming\nltk_data…
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\wajah\AppData\Roaming\nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
```

[80]:
```
True
```

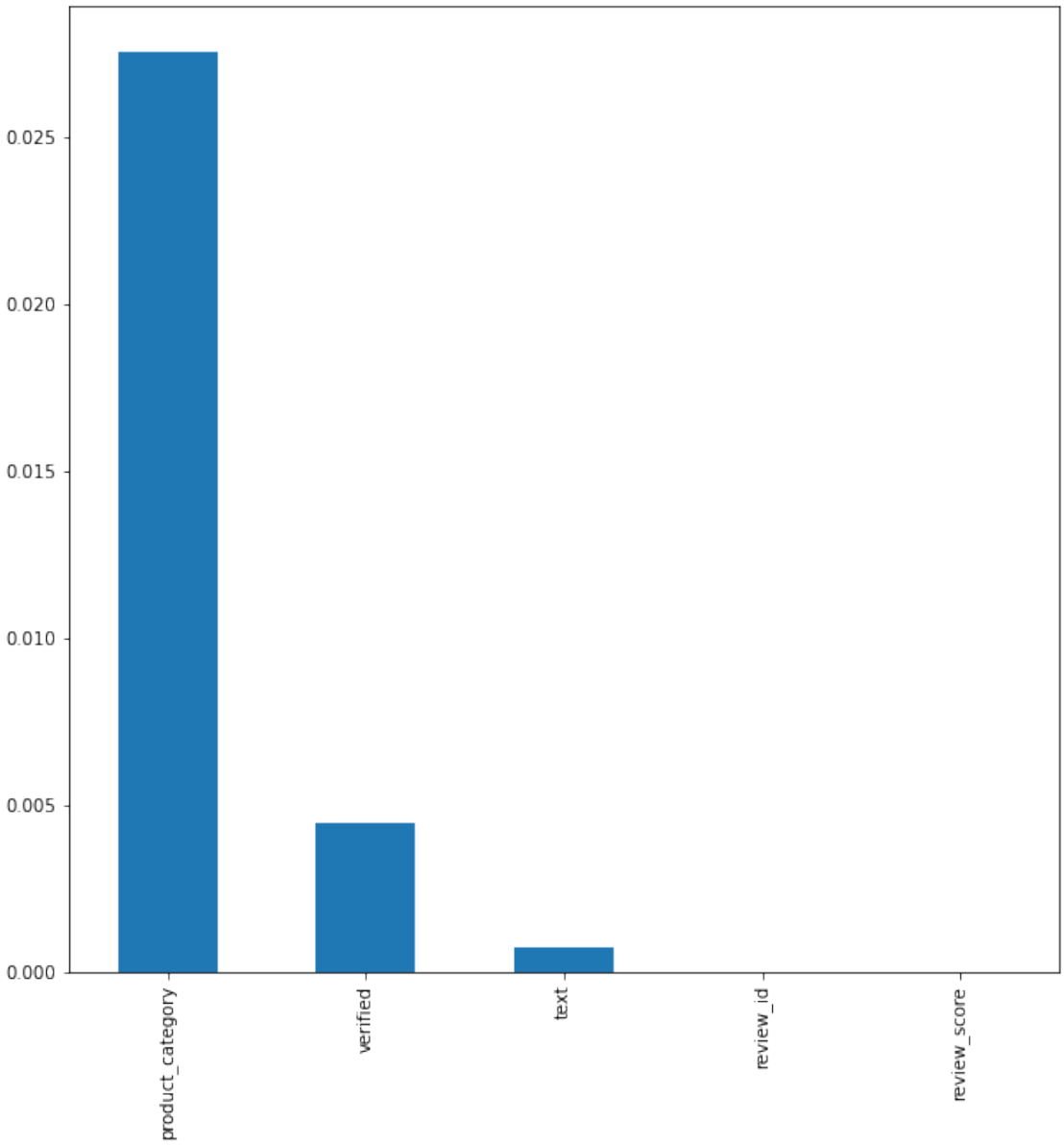
[81]:
```
#reading dataset
df = pd.read_csv("processed_reviews_split_RESIT_minimal.csv")
df.head()
```

[81]:
```
                review_id                                               text  \
0  product_review_000000                  OMG this is sooo good.. Very Good!!!
1  product_review_000001  This soap smells pretty good and it seems to w…
2  product_review_000002  Don't seem to dissolve after quite some time. …
3  product_review_000003  these zip bags do as they should. very good. I…
4  product_review_000004  What. A. Mess.\nI recommend making this in a m…

  verified  review_score product_category
0     True           5.0     prime_pantry
1     True           4.0     prime_pantry
2    False           1.0     prime_pantry
3     True           5.0     prime_pantry
4     True           2.0     prime_pantry
```

[82]:
```
#checking null values in percentage.
percent_missing = df.isnull().sum() * 100 / len(df)
missing_value_df = pd.DataFrame({'column_name': df.columns,'percent_missing':␣
 ↪percent_missing})
```

[83]:
```
#ploting numer of missing values in different columns in percentage.
import matplotlib.pyplot as plt
Miss_val = df.isna().sum()/df.shape[0]
Miss_val.sort_values(ascending=False, inplace=True)
Miss_val
plt.figure(figsize=(10,10))
Miss_val.plot.bar()
```

[83]:
```
<AxesSubplot:>
```



[84]:
```
drp=df[df.isnull().any(axis=1)]
idd=drp['review_id']
idd=list(idd)
df['review_id']
```

[84]:
```
0         product_review_000000
1         product_review_000001
2         product_review_000002
3         product_review_000003
4         product_review_000004
                  ...
```

```
28148    product_review_028148
28149    product_review_028149
28150    product_review_028150
28151    product_review_028151
28152    product_review_028152
Name: review_id, Length: 28153, dtype: object
```

[85]:
```
l=[]
for i in df['review_id']:

    if i in idd:
        exc=i+",1,missing_value_or_label"
        l.append(exc)
    else:
        exc=i+",0,N/A"
        l.append(exc)
```

[86]:
```
exclusivedf=df.copy()
exclusivedf = exclusivedf.drop('text', axis = 1)
exclusivedf['reason_for_exclusion']=l
```

[87]:
```
exclusivedf.to_csv('exclusions_resit_dataset_problem2.csv')
```

[88]:
```
#removing rows with missing values
df = df.dropna(axis=0)
df=df.reset_index(drop=True)
df
```

[88]:
```
                    review_id  \
0      product_review_000000
1      product_review_000001
2      product_review_000002
3      product_review_000003
4      product_review_000004
…                        …
27228  product_review_028147
27229  product_review_028148
27230  product_review_028149
27231  product_review_028151
27232  product_review_028152

                                                  text verified  \
0                    OMG this is sooo good.. Very Good!!!     True
1      This soap smells pretty good and it seems to w…     True
2      Don't seem to dissolve after quite some time. …    False
3      these zip bags do as they should. very good. I…     True
4      What. A. Mess.\nI recommend making this in a m…     True
```

```
…                                                     …      …
27228  Ordered two of the bandaids which we use most…     True
27229                                    Taste is ok..      True
27230                            healthy tasty side dish.   True
27231  Don't let the packaging turn you away - Jack B…     False
27232  I am quite light-skinned and when I was asked …     False

       review_score product_category
0               5.0     prime_pantry
1               4.0     prime_pantry
2               1.0     prime_pantry
3               5.0     prime_pantry
4               2.0     prime_pantry
…               …               …
27228           5.0     prime_pantry
27229           5.0     prime_pantry
27230           5.0     prime_pantry
27231           5.0    luxury_beauty
27232           5.0    luxury_beauty

[27233 rows x 5 columns]
```

1 Data Cleaning

2 Removing unwanted spaces

[89]:
```
def remove_spaces(text):
    text=text.strip()
    text=text.split()
    return ' '.join(text)
```

3 Contraction

[90]:
```
contraction = {'cause':'because', 'aint': 'am not', 'aren\'t': 'are not'}

def mapping_replacer(x,dic):
    for words in dic.keys():
        if ' ' + words + ' ' in x:
            x=x.replace(' '+ words +' ' ,' '+dic[words]+' ' )
    return x
```

[91]:
```
from nltk.tokenize import word_tokenize
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.stem.lancaster import LancasterStemmer
```

```
nltk.LancasterStemmer
ls = LancasterStemmer()
lem = WordNetLemmatizer()
def lexicon_normalization(text):
    words = word_tokenize(text)


    #Stemming
    words_stem = [ls.stem(w) for w in words]

    #Lemmatization
    words_lem = [lem.lemmatize(w) for w in words_stem]
    return words_lem
```

4 Removing hyperlinks, numbers, punctuations, brackets etc.

[92]:
```
# updating text to lowercase, removing text between square brackets, Remove␣
 ↪hyperlinks, Remove the punctuation and remove words that are containing␣
 ↪numbers.
def clean_text(text):
    text = str(text).lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('_', '', text)
    text = re.sub('\n', '', text)
    text = re.sub('\w*\d\w*', '', text)
    text = re.sub('\'','', text)
    return text
```

5 Managing stopwords

[93]:
```
from collections import Counter
def remove_stopword(text):
    stop_words = stopwords.words('english')
    stopwords_dict = Counter(stop_words)
    text = ' '.join([word for word in text.split() if word not in␣
 ↪stopwords_dict])
    return text
```

6 Tokenisation

[94]:
```
def tokenise(text):
    words = word_tokenize(text)
    return words
```

7 Handling Regular expressions

[95]:
```
import re
df['text'] = df['text'].map(lambda x: re.sub(r'\W+', ' ', str(x)))
df['text'] = df['text'].replace(r'\W+', ' ', regex=True)
```

[96]:
```
df['text']=df['text'].apply(lambda x: mapping_replacer(x, contraction))
```

[97]:
```
df['text']=df['text'].apply(lambda x:clean_text(x))
```

[98]:
```
df['text']=df['text'].apply(lambda x: remove_stopword(x))
```

[99]:
```
df['text']=df['text'].apply(lambda x: lexicon_normalization(x))
```

8 Finding Common words in text column of dataset

9 Common words for Luxury Beauty products

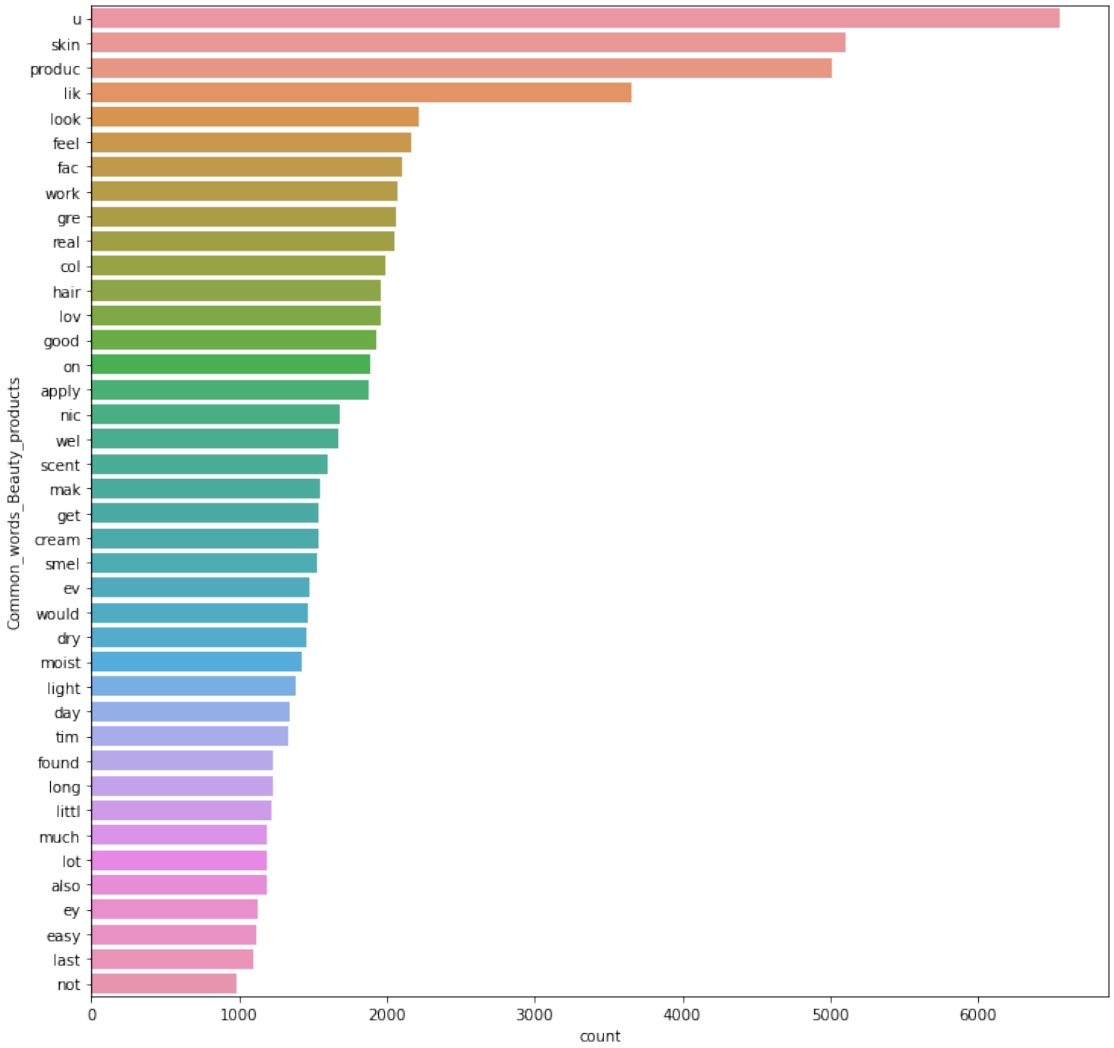
[100]:
```
beauty=df[df['product_category']=='luxury_beauty']
```

[101]:
```
top = Counter([item for sublist in beauty['text'] for item in sublist])
temp = pd.DataFrame(top.most_common(40))
temp.columns = ['Common_words_Beauty_products','count']
temp.style.background_gradient(cmap='Oranges')
```

[101]:
```
<pandas.io.formats.style.Styler at 0x155a343fbe0>
```

[102]:
```
import seaborn as sns
plt.figure(figsize=(12,12))
sns.barplot(temp['count'],temp['Common_words_Beauty_products'])
```

[102]:
```
<AxesSubplot:xlabel='count', ylabel='Common_words_Beauty_products'>
```



10 Common words for Prime Pantry products

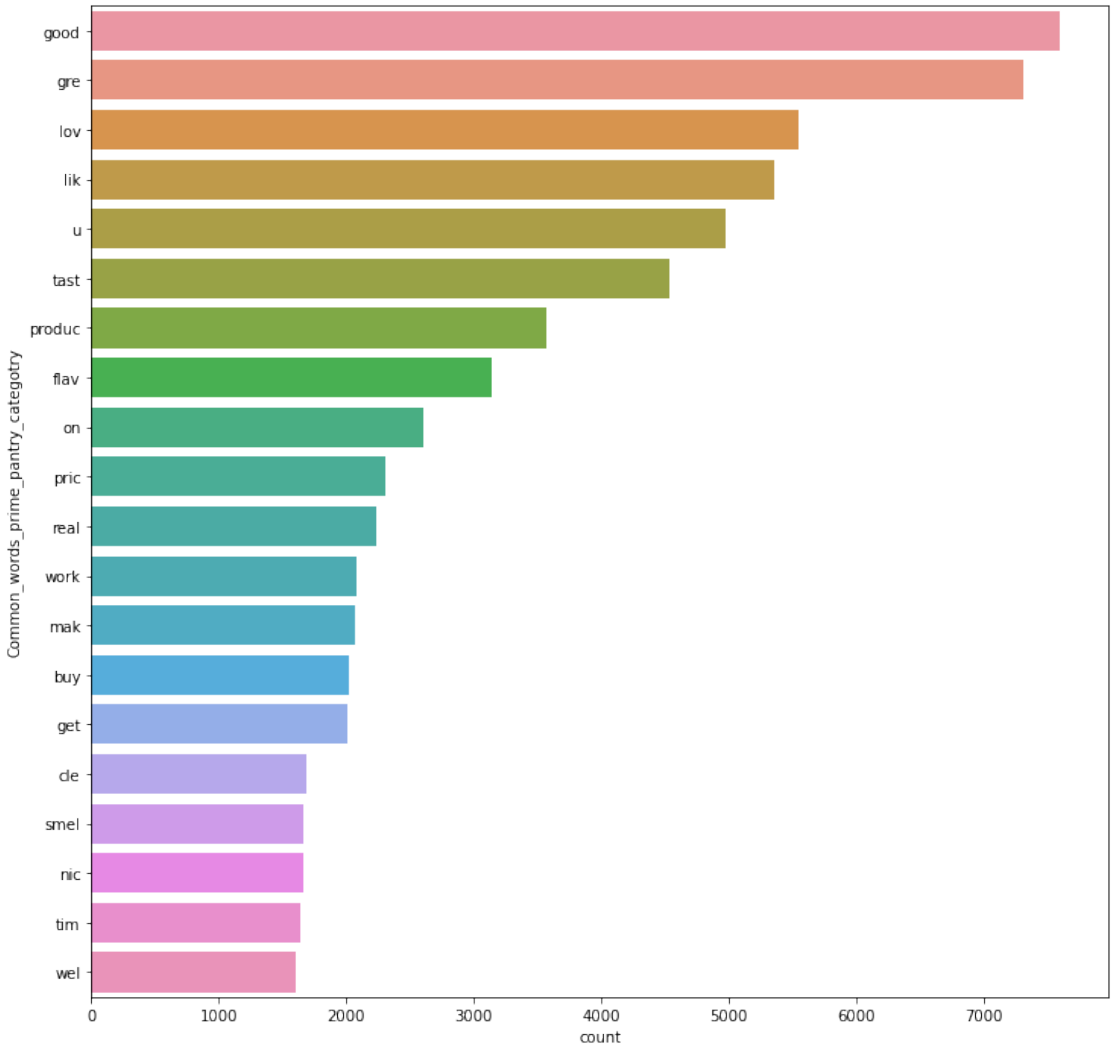
[103]:
```
pantry=df[df['product_category']=='prime_pantry']
```

[104]:
```
top2 = Counter([item for sublist in pantry['text'] for item in sublist])
temp2 = pd.DataFrame(top2.most_common(20))
temp2.columns = ['Common_words_prime_pantry_categotry','count']
temp2.style.background_gradient(cmap='Oranges')
```

[104]:
```
<pandas.io.formats.style.Styler at 0x155b1165b50>
```

[105]:
```
plt.figure(figsize=(12,12))
sns.barplot(temp2['count'],temp2['Common_words_prime_pantry_categotry'])
```

[105]: <AxesSubplot:xlabel='count', ylabel='Common_words_prime_pantry_categotry'>



[106]:
```
l=[]
for i in df['text']:
    str1 = " "
    t=str1.join(i)
    l.append(t)
df['text']=l
```

11 Predicting Product Categories

[107]:
```
 # using Bag of words technique for Converting feature into a vector
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df['text'])
b=vectorizer.get_feature_names()
```

[108]:
```
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from sklearn.linear_model import LogisticRegression
df
```

[108]:

	review_id	text	verified	review_score	product_category
0	product_review_000000	omg sooo good good	True	5.0	prime_pantry
1	product_review_000001	soap smel pretty good seem work gre definit mu…	True	4.0	prime_pantry
2	product_review_000002	seem dissolv quit tim see cle benefit smel nic…	False	1.0	prime_pantry
3	product_review_000003	zip bag good happy awesom	True	5.0	prime_pantry
4	product_review_000004	mess recommend mak much larg contain littl bow…	True	2.0	prime_pantry
…	…	…	…	…	…
27228	product_review_028147	ord two bandaid u band aid brand adher band to…	True	5.0	prime_pantry
27229	product_review_028148	tast ok	True	5.0	prime_pantry
27230	product_review_028149	healthy tasty sid dish	True		
27231	product_review_028151	let pack turn away jack black ep moist cle cre…	False		
27232	product_review_028152	quit light skin ask try shad thought might dar…	False		

```
27230           5.0     prime_pantry
27231           5.0    luxury_beauty
27232           5.0    luxury_beauty

[27233 rows x 5 columns]
```

[109]:
```
ddf=df.drop(['text','review_id'],axis=1)
```

[110]:
```
 #creating an dataframe and converted it to a vector
df1 = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names())
res = pd.concat([df1, ddf], axis=1)
```

12 Encoding by categories.

[111]:
```
a=df.select_dtypes(include='object')
```

[112]:
```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
df['product_category']=le.fit_transform(df['product_category'])
```

[113]:
```
# Assigning inputs and outputs as X and y
X=df1
y=df['product_category']
```

[114]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import recall_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
```

[115]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20,␣
 ↪random_state=42)
```

[116]:
```
from sklearn.tree import DecisionTreeClassifier
import seaborn as sns
from sklearn.metrics import confusion_matrix
```

[117]:
```
dt =DecisionTreeClassifier(random_state=10000)
```

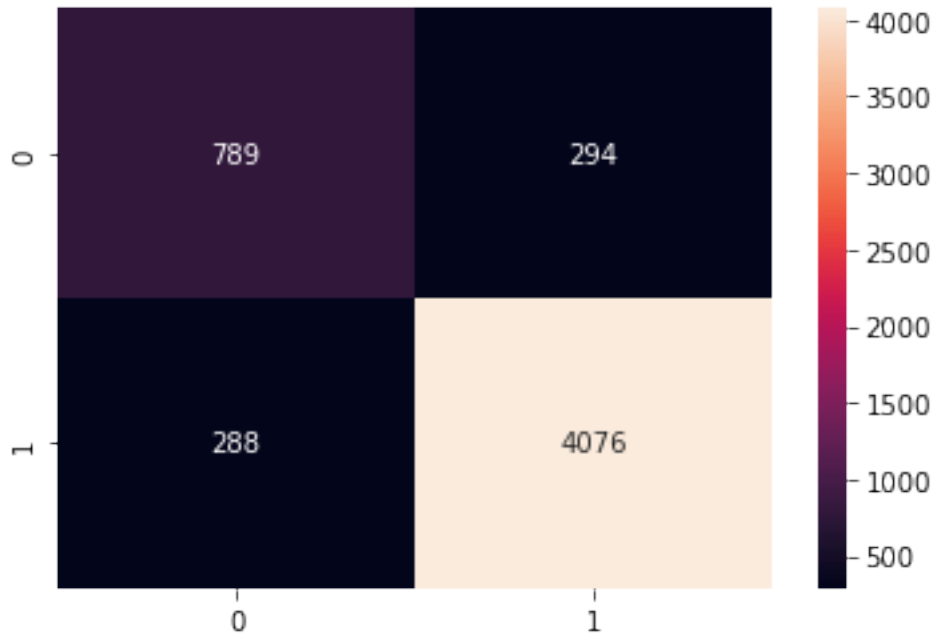
[118]:
```
dt.fit(X_train, y_train)
```

[118]:
```
DecisionTreeClassifier(random_state=10000)
```

[119]:
```
y_predict_dt = dt.predict(X_test)
y_predict_dt_train = dt.predict(X_train)
# confusion_matrix
cm = confusion_matrix(y_test, y_predict_dt)
```

```
sns.heatmap(cm, annot=True, fmt="d")
```

[119]: <AxesSubplot:>



[120]:
```
print('train accuracy',accuracy_score((y_train), y_predict_dt_train))
print('test accuracy',accuracy_score((y_test), y_predict_dt))
print('precision',precision_score(y_test, y_predict_dt, average='macro'))
print('recall',recall_score(y_test, y_predict_dt, average='macro'))
```

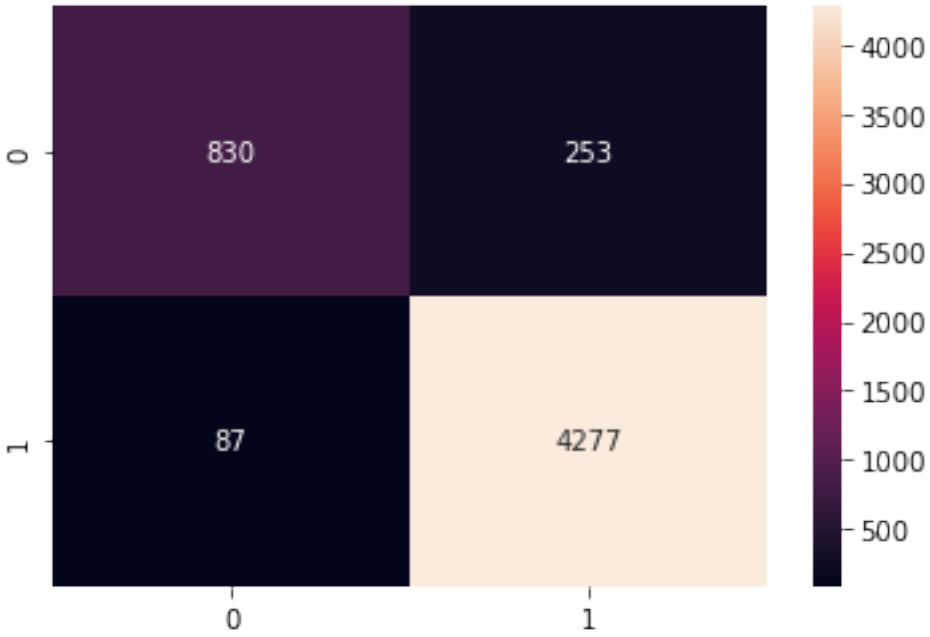
```
train accuracy 0.9972000367208299
test accuracy 0.8931521938681843
precision 0.8326568206880287
recall 0.8312686777486917
```

[121]:
```
print(classification_report(y_test, y_predict_dt))
```

```
              precision    recall  f1-score   support

           0       0.73      0.73      0.73      1083
           1       0.93      0.93      0.93      4364

    accuracy                           0.89      5447
   macro avg       0.83      0.83      0.83      5447
weighted avg       0.89      0.89      0.89      5447
```

[122]:
```
from sklearn.ensemble import RandomForestClassifier
rf =RandomForestClassifier(random_state=10000)
rf.fit(X_train, y_train)
y_predict_rf = rf.predict(X_test)
y_predict_rf_train = rf.predict(X_train)
# confusion_matrix
cm = confusion_matrix(y_test, y_predict_rf)
sns.heatmap(cm, annot=True, fmt="d")
```

[122]: <AxesSubplot:>



[123]:
```
print('train accuracy',accuracy_score((y_train), y_predict_rf_train))
print('test accuracy',accuracy_score((y_test), y_predict_rf))
print('precision',precision_score(y_test, y_predict_rf))
print('recall',recall_score(y_test, y_predict_rf))
```

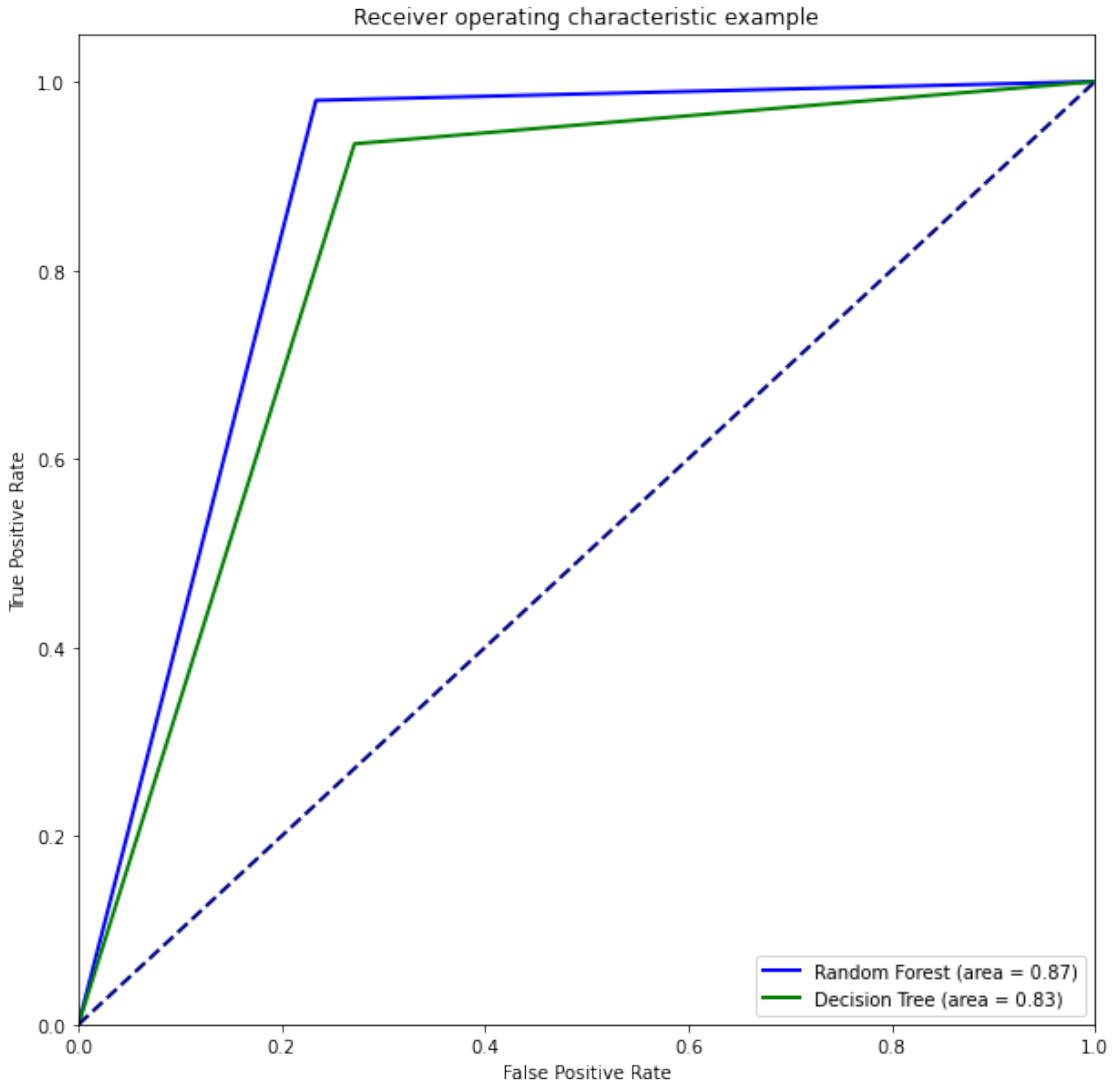
```
train accuracy 0.9972000367208299
test accuracy 0.9375803194418946
precision 0.944150110375276
recall 0.98006416131989
```

[124]:
```
print(classification_report(y_test, y_predict_rf))
```

```
              precision    recall  f1-score   support

           0       0.91      0.77      0.83      1083
```

```
              1       0.94      0.98      0.96      4364

    accuracy                            0.94      5447
   macro avg       0.92      0.87      0.90      5447
weighted avg       0.94      0.94      0.94      5447
```

[125]:
```
### calculating ROC curve and area for predicting over validation set.
from sklearn.metrics import roc_curve, auc

### Ploting
plt.figure(figsize=(10,10))
lw = 2
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_predict_rf)
roc_auc_rf = auc(fpr_rf, tpr_rf)
plt.plot(fpr_rf, tpr_rf, color='blue',
         lw=lw, label='Random Forest (area = %0.2f)' % roc_auc_rf)
fpr_dt, tpr_dt, _ = roc_curve(y_test, y_predict_dt)
roc_auc_dt = auc(fpr_dt, tpr_dt)
plt.plot(fpr_dt, tpr_dt, color='green',
         lw=lw, label='Decision Tree (area = %0.2f)' % roc_auc_dt)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```



13 Predicting Review Scores

[126]:
```
vectorizer2 = CountVectorizer(max_features=2500)
X2 = vectorizer2.fit_transform(df['text'])
b2=vectorizer2.get_feature_names()
```

[127]:
```
#defining dataframe which is converted into vector which also contains other␣
↪features
df2 = pd.DataFrame(X2.toarray(), columns=vectorizer2.get_feature_names())
```

[128]:
```
X2=df2
y2=df[['review_score']]
```

[129]:
```
from collections import Counter
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
# summarizeing the class distribution
print(y2.value_counts())
# defining resampling
# transforming the given dataset
strategy = {5.0:18681, 4.0:10000,3.0:10000,2.0:9000,1.0:8500,-1.0:8000}
over = RandomOverSampler(sampling_strategy=strategy)
# defining pipeline
pipeline = Pipeline(steps=[('o', over)])
X2, y2 = pipeline.fit_resample(X2, y2)
# summarize class distribution
print(y2.value_counts())
```

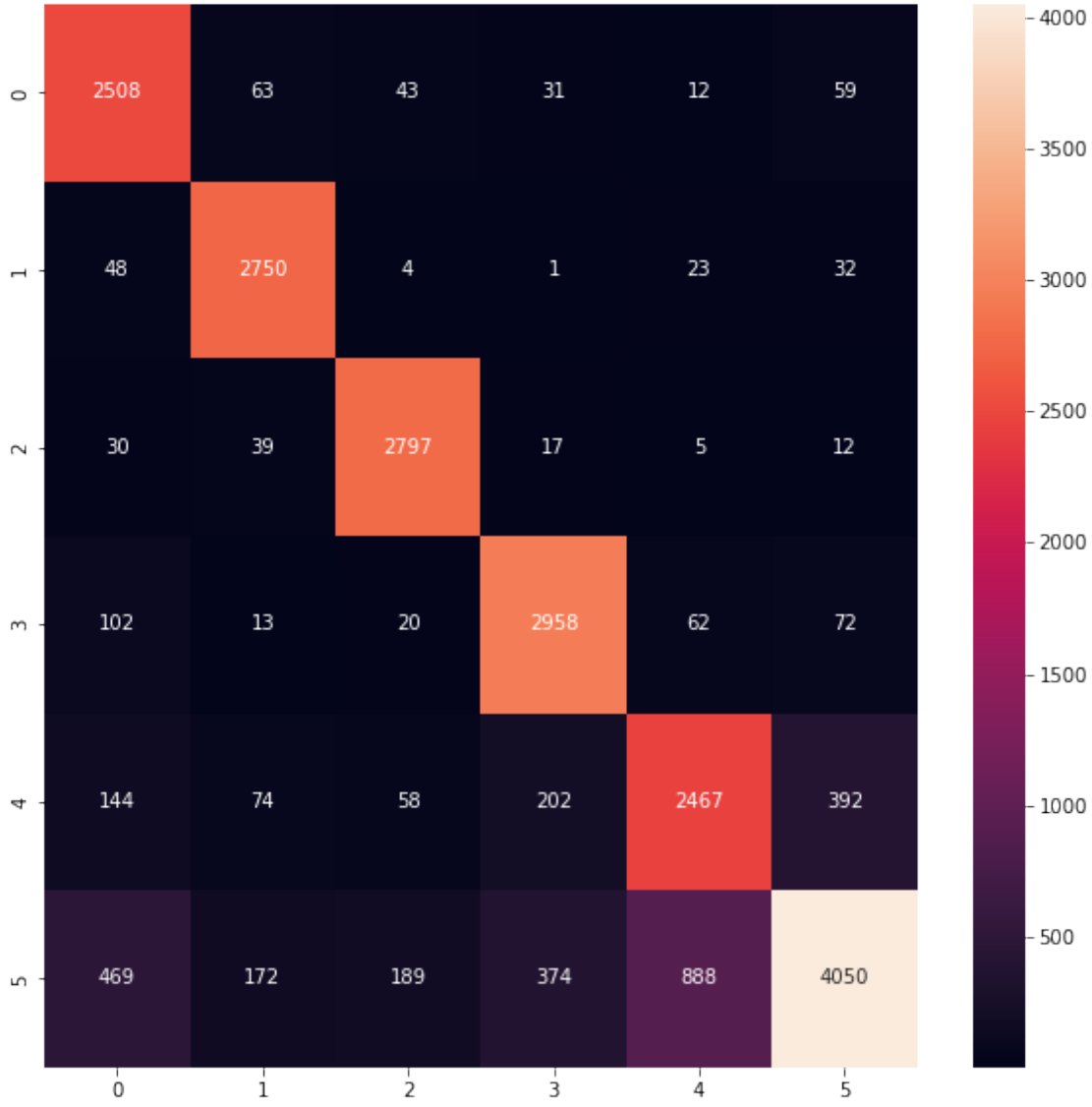
```
review_score
 5.0            17790
 4.0             4678
 3.0             2301
 2.0              938
-1.0              769
 1.0              757
dtype: int64
review_score
 5.0            18681
 3.0            10000
 4.0            10000
 2.0             9000
 1.0             8500
-1.0             8000
dtype: int64
```

[130]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X2 = scaler.fit_transform(X2)
```

[131]:
```
X_train2, X_test2, y_train2, y_test2 = train_test_split(X2,y2, test_size=0.33,␣
↪random_state=42)
```

[132]:
```
dt2 =DecisionTreeClassifier(random_state=1024)
dt2.fit(X_train2, y_train2)
y_predict_dt2 = dt2.predict(X_test2)
# confusion_matrix
cm = confusion_matrix(y_test2, y_predict_dt2)
plt.figure(figsize=(10,10))
sns.heatmap(cm, annot=True, fmt="d")
```

[132]: <AxesSubplot:>



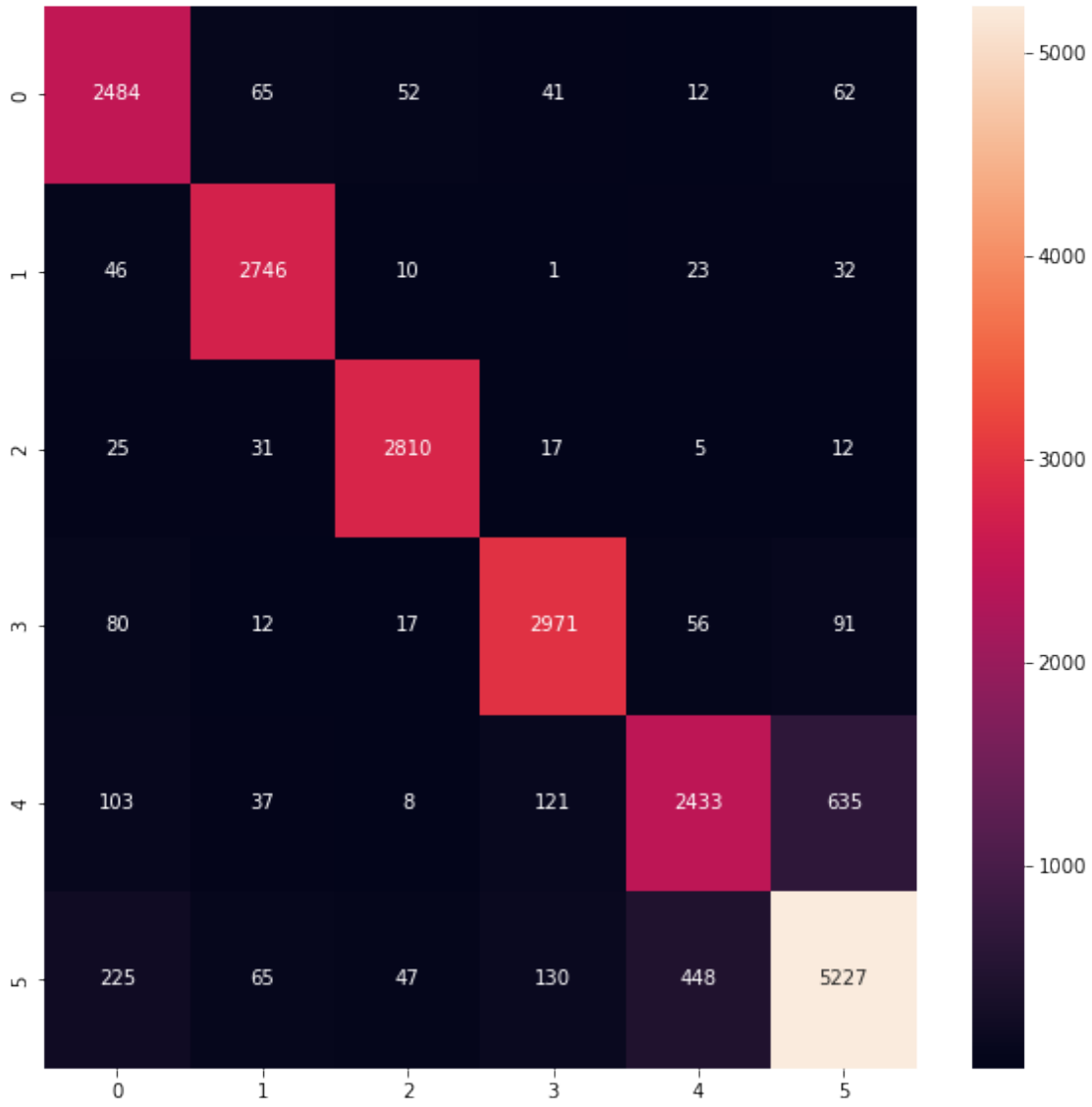
[133]:
```
print('accuracy',accuracy_score((y_test2), y_predict_dt2))
print('precision',precision_score(y_test2, y_predict_dt2, average='macro'))
print('recall',recall_score(y_test2, y_predict_dt2, average='macro'))
```

```
accuracy 0.82766761095373
precision 0.8265300688783562
recall 0.8609054743556918
```

[134]:
```
rf2 =RandomForestClassifier(random_state=1000)
rf2.fit(X_train2, y_train2)
```

```
y_predict_rf2 = rf2.predict(X_test2)
# confusion_matrix
cm = confusion_matrix(y_test2, y_predict_rf2)
plt.figure(figsize=(10,10))
sns.heatmap(cm, annot=True, fmt="d")
```

[134]: <AxesSubplot:>



[135]:
```
print('accuracy',accuracy_score((y_test2), y_predict_rf2))
print('precision',precision_score(y_test2, y_predict_rf2, average='macro'))
print('recall',recall_score(y_test2, y_predict_rf2, average='macro'))
```

```
accuracy 0.8815391879131256
```

```
precision 0.8845411989173232
recall 0.8908584346463453
```

[ ]:

[ ]:

[ ]: