Sentiment Analysis for Customer Reviews Challenge

Challenge:

Develop a robust Sentiment Analysis classifier for XYZ customer reviews, automating the categorization into positive, negative, or neutral sentiments. Utilize Natural Language Processing (NLP) techniques, exploring different sentiment analysis methods.

Problem Statement:

XYZ organization, a global online retail giant, accumulates a vast number of customer reviews daily. Extracting sentiments from these reviews offers insights into customer satisfaction, product quality, and market trends. The challenge is to create an effective sentiment analysis model that accurately classifies XYZ customer reviews.

Important Instructions:

1. **Make sure this ipynb file that you have cloned is in the Project folder on the Desktop. The Dataset is also available in the same folder.**
2. **Ensure that all the cells in the notebook can be executed without any errors.**
3. **Once the Challenge has been completed, save the SentimentAnalysis.ipynb notebook in the *Project* Folder on the desktop. If the file is not present in that folder, autoevolution will fail.**
4. **Print the evaluation metrics of the model.**
5. **Before you submit the challenge for evaluation, please make sure you have assigned the Accuracy score of the model that was created for evaluation.**
6. **Assign the Accuracy score obtained for the model created in this challenge to the specified variable in the predefined function *submit_accuracy_score*. The solution is to be written between the comments `# code starts here` and `# code ends here`**

7. **Please do not make any changes to the variable names and the function name *submit_accuracy_score* as this will be used for automated evaluation of the challenge. Any modification in these names will result in unexpected behaviour.**

Data Sources:

The dataset for this sentiment analysis project consists of XYZ customer reviews. The data contains the following columns:

- Id: Row identifier.
- ProductId: Unique identifier for the product.
- UserId: Unique identifier for the user.
- ProfileName: Profile name of the user.
- HelpfulnessNumerator: Number of users who found the review helpful.
- HelpfulnessDenominator: Number of users who indicated whether they found the review helpful or not.
- Score: Rating between 1 and 5.
- Time: Timestamp for the review.
- Summary: Brief summary of the review.
- Text: Text of the review.

The data was collected using a combination of publicly available datasets and web scraping methods. Ethical considerations were taken into account during data collection to ensure compliance with privacy standards.

Preprocessing Steps:

Text Cleaning:
- Removed irrelevant information, HTML tags, and special characters from the text data.

Tokenization:
- Broke down sentences into individual words to facilitate analysis.

Handling Missing Values:
- Addressed missing values through imputation or data removal.

Lowercasing:
- Converted all text to lowercase for uniformity in analysis.

Model Architectures:

Feature Extraction:

- Utilized TF-IDF (Term Frequency-Inverse Document Frequency) for converting text data into numerical features.

Model Selection:

- Experimented with various models, including Naive Bayes,

Training and Testing:

- Split the dataset into training and testing sets for model evaluation.

Evaluation Metrics:

- Utilized accuracy, precision, recall, and F1 score to assess model performance.

Deployment Procedures:

Model Integration:

- Choose AWS as the deployment environment to host the sentiment analysis model.

Real-time Predictions:

- Configured the model for making real-time predictions on new XYZ customer reviews, ensuring efficiency and accuracy.

Monitoring:

- Implemented monitoring tools to track the model's performance in a production environment.

Continuous Improvement:

- Developed a plan for updating the model to adapt to changing trends and language usage, ensuring its relevancy over time.

Code Documentation:

Sample Code:

```
# Data collection
# source 1 : Reviews.csv file provided
```

In [7]:

```
# Data preprocessing

# reading csv using pandas
import pandas as pd
df = pd.read_csv('Reviews.csv')
```

```
print(df.head())
```

```
   Id   ProductId          UserId                      ProfileName  \
0   1  B001E4KFG0  A3SGXH7AUHU8GW                       delmartian
1   2  B00813GRG4  A1D87F6ZCVE5NK                        dll pa
2   3  B000LQOCH0   ABXLMWJIXXAIN  Natalia Corres "Natalia Corres"
3   4  B000UA0QIQ  A395BORC6FGVXV                         Karl
4   5  B006K2ZZ7K  A1UQRSCLF8GW1T    Michael D. Bigham "M. Wassir"

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     1                       1      5  1303862400
1                     0                       0      1  1346976000
2                     1                       1      4  1219017600
3                     3                       3      2  1307923200
4                     0                       0      5  1350777600

                Summary                                               Text
0  Good Quality Dog Food  I have bought several of the Vitality canned d...
1      Not as Advertised  Product arrived labeled as Jumbo Salted Peanut...
2  "Delight" says it all  This is a confection that has been around a fe...
3         Cough Medicine  If you are looking for the secret ingredient i...
4            Great taffy  Great taffy at a great price.  There was a wid...
```

In [8]:

```
# text cleaning
# 1. removing leading and trailing spaces
df["Summary"] = df["Summary"].str.strip()
df["Text"] = df["Text"].str.strip()

# 2. Removing special characters
df["Summary"] = df["Summary"].str.replace("[\"$&+,:;=?@#|'<>.-^*()%!]", "")
df["Text"] = df["Text"].str.replace("[\"$&+,:;=?@#|'<>.-^*()%!]", "")

# tokenization
#import nltk
#df["tokenized summary"] = df.apply(lambda row: nltk.word_tokenize(row["Summary"]), axis=1)
#df["tokenized text"] = df.apply(lambda row: nltk.word_tokenize(row["Text"]), axis=1)

# Handling missing values
# no missing values
df = df.dropna()

# Lowercasing
df["Summary"] = df["Summary"].str.lower()
```

```
df["Text"] = df["Text"].str.lower()

print(df.head())
print(df.index)
```

```
   Id   ProductId          UserId                      ProfileName  \
0   1  B001E4KFG0  A3SGXH7AUHU8GW                       delmartian
1   2  B00813GRG4  A1D87F6ZCVE5NK                        dll pa
2   3  B000LQOCH0   ABXLMWJIXXAIN  Natalia Corres "Natalia Corres"
3   4  B000UA0QIQ  A395BORC6FGVXV                          Karl
4   5  B006K2ZZ7K  A1UQRSCLF8GW1T    Michael D. Bigham "M. Wassir"

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     1                       1      5  1303862400
1                     0                       0      1  1346976000
2                     1                       1      4  1219017600
3                     3                       3      2  1307923200
4                     0                       0      5  1350777600

                Summary                                               Text
0  good quality dog food  i have bought several of the vitality canned d...
1      not as advertised  product arrived labeled as jumbo salted peanut...
2  "delight" says it all  this is a confection that has been around a fe...
3         cough medicine  if you are looking for the secret ingredient i...
4            great taffy  great taffy at a great price.  there was a wid...
Index([     0,      1,      2,      3,      4,      5,      6,      7,      8,
           9,
       ...
       568444, 568445, 568446, 568447, 568448, 568449, 568450, 568451, 568452,
       568453],
      dtype='int64', length=568401)
```

In [10]:

```
# sentiment analysis implementation
# 1. Naive bayes algorithm
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score
from sklearn.pipeline import make_pipeline

# Map scores to sentiments (e.g., positive, neutral, negative)
df['Sentiment'] = df['Score'].apply(lambda score: 'positive' if score > 3 else ('negative' if score < 3
else 'neutral'))
```

```
# Split the data into training and testing sets
train_data, test_data = train_test_split(df, test_size=0.2, random_state=42)

# Use TF-IDF vectorizer
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)

# Create TF-IDF matrices for training and testing data
X_train = vectorizer.fit_transform(train_data['Summary'])
X_test = vectorizer.transform(test_data['Summary'])

# Use a simple model (Naive Bayes) as a starting point
model = make_pipeline(MultinomialNB())
model.fit(X_train, train_data['Sentiment'])

# Make predictions on the test set
predictions = model.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(test_data['Sentiment'], predictions))
print("\nClassification Report:\n", classification_report(test_data['Sentiment'], predictions))
```

Accuracy: 0.8238315989479332

Classification Report:

	precision	recall	f1-score	support
negative	0.84	0.33	0.47	16452
neutral	0.63	0.01	0.01	8460
positive	0.82	0.99	0.90	88769
accuracy			0.82	113681
macro avg	0.76	0.44	0.46	113681
weighted avg	0.81	0.82	0.77	113681

In [13]:

```
from sklearn.svm import SVC
svm_model = SVC()
X_train, X_test, y_train, y_test = train_test_split(df['Summary'],
                    df['Sentiment'], test_size=0.2, random_state=42)
# Create TF-IDF matrices for training and testing data
X_train = vectorizer.fit_transform(train_data['Summary'])
X_test = vectorizer.transform(test_data['Summary'])

svm_model.fit(X_train, y_train)
```

```
svm_predictions = svm_model.predict(X_test)
```

In [ ]:

```
# Evaluate SVM
print("SVM Accuracy:", accuracy_score(y_test, svm_predictions))
print("\nClassification Report:\n", classification_report(y_test, svm_predictions))
```

---------- CHALLENGE CODE ENDS HERE ----------

NOTE:

1. Assign the Accuracy score obtained for the model created in this challenge to the specified variable in the predefined function *submit_accuracy_score* below. The solution is to be written between the comments `# code starts here` and `# code ends here`
2. Please do not make any changes to the variable names and the function name *submit_accuracy_score* as this will be used for automated evaluation of the challenge. Any modification in these names will result in unexpected behavior.

In [ ]:

```
def submit_accuracy_score()-> float:
    #accuracy should be in the range of 0.0 to 1.0
    accuracy = 0.0
    # code starts here

    # code ends here
     return accuracy
```