Twitter Sentiment Analysis

A Text Classification Project

1 Introduction

1.1 Introduction to the Problem area

In this text classification project, the main focus revolves around sentiment analysis of a Tweets dataset. The data was acquired from the Kaggle website link. The dataset is an entity-level sentiment analysis dataset of Twitter, it contains the following classes: positive, negative, neutral and irrelevant, a training set and a validation set which facilitates the process of developing and testing the classification models.

The chosen field, sentiment analysis, has witnessed major growth and improvements, with extensive research and a diverse range of approachs and methodologies. This project aims to showcase ambitious and innovative approaches and insights by building upon the existing knowledge and resources of this rapidly evolving field.

The dataset has multiple associated notebooks on Kaggle [1] which was used as a measure to compare the classification results of this project. Furthermore, great efforts were made to ensure the uniqueness of the approach and methods applied in the project and the expected contribution of this project lies in its ability to:

- Provide comprehensive evaluation and comparison of various classification algorithms in the context of sentiment analysis on Twitter data.
- Evaluate the influence and impact of using different feature representations on model performance.
- Provide a thorough understanding of the advantages and disadvantages of each model to help practitioners select the best strategies for similar projects/tasks.
- Present a refined best-performing model showcasing the potential improvements achievable by optimization.

While sentiment analysis on social media data is a popular and well-explored area, this project's originality lies in its comprehensive approach to evaluating multiple algorithms, optimizing a specific model, and providing nuanced insights. The ambition and novelty stem from the combination of methodological diligence, diverse model evaluation metrics, and the exploration of hyperparameter optimization techniques, collectively contributing to advancements in sentiment analysis.

1.2 Objectives

This project's main objective is to deploy advanced text classification techniques to tackle the challenges posed by sentiment analysis in Twitter data. The objectives of this project are as follows:

- **Data preprocessing and cleaning:** To ensure data usability in our text classification project, robust data preprocessing and cleaning techniques have been applied to our data.

- **Exploratory Data Analysis (EDA):** Conduct an EDA in order to better understand and gain insight into the data.

- **Text Representation:** Apply different vectorization techniques (Count vectorizer, TF-IDF) to convert data into numerical features suitable for machine learning models. Evaluate and compare the performance of each vectorization method across multiple classification algorithms.

- **Model Implementation:** Apply logistic regression and Naive Bayes as baselines, SVM, random forest and KNN classification algorithms to the vectorized data. Compare and evaluate the performance of each model in handling Twitter sentiments.

- **Hyperparameter Tuning:** Employ RandomizedSearchCV to further optimize hyperparameters for the best-performing model.

- **Cross-validation:** Implement cross-validation techniques to asses the generalization capability of the best-performing model. The latter ensures consistency of model performance across different subsets of the data.

- **Confusion Matrix Analysis:** To further evaluate model performance, develop confusion matrices for each classification model, providing a detailed breakdown of true positives, true negatives, false positives, and false negatives.

1.3 Dataset

The 'Twitter sentiment analysis' dataset was obtained from the Kaggle website link.
The dataset contains two CSV files, a training set and a validation set, the training set size is 10.33 MB and its shape is 74682 rows by 4 columns and the validation set size is 164.35 kB and its shape is 1000 rows by 4 columns. There is a column containing an ID for each tweet and is of type 'int64', a column of type object with the name of the company the tweet talks about, a column with type object containing the label/sentiment and finally, a column with type object containing the tweet text. Moreover, The columns didn't have any headers so the headers were added manually in the preprocessing and cleaning process.

Furthermore, this dataset was chosen for this text classification project due to the rise of social media sentiment analysis which spiked up an interest to explore this evolving field aspiring to discover new insights and optimized methodologies. The data chosen is appropriate for this project as it contains tweets that belong to different classes/sentiments (positive, negative, ..etc), where we can use text classification

techniques to predict/classify tweet classes, training on the training set and validating our results using the validation set.

1.4 Evaluation Methodology

One of the simplest and most widely used metrics for sentiment analysis is accuracy. However, using accuracy alone can be extremely misleading especially if the data is imbalanced or skewed. For instance, if a model that always predicts true was applied on data that contains 90% true labels and 10% false would get an accuracy of 90%, which can be tremendously misleading.

Therefore, a wide range of metrics was used to evaluate the performance of models, by using a classification report which provides a breakdown of how the model performs on each class by presenting precision, recall, F1-score, support, accuracy, macro average and weighted average.

- Precision: refers to the number of true positives divided by the total number of positive predictions[2] and it indicates the quality of positive prediction produced by the model.
- Recall: is the true positive rate (TRT), it refers to the percentage of correctly identified data samples[3].
- F1-score: F1 score is a measure of the harmonic mean of precision and recall[4]. It combines precision and recall into one metric to better understand model performance.
- Support: the number of samples of the true response that lies in each class of target values[5].
- Accuracy: measures the number of correct predictions made by the model. It's calculated by dividing correct predictions by the total predictions made.
- Macro-average: calculates average performance across all classes.
- Weighted-average: it's the precision of all classes together, the total number TP(true positive of all classes) divided by the total number of objects in all classes.

Furthermore, cross-validation will be used with the best overall performing model after optimization to measure how well the model performs and to ensure stability and consistent performance.

In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing
import matplotlib.pyplot as plt #Plotting properties
import seaborn as sns
# Set style for seaborn for better readability
sns.set(style="whitegrid")
from wordcloud import WordCloud

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
from sklearn.model_selection import cross_val_score, StratifiedKFold

from gensim.models import Word2Vec

import nltk
from nltk import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt')

import re #Regular expressions
import emoji
import time

# Show all matplotlib graphs inline
%matplotlib inline
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\user\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\user\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\user\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

2 Implementation

2.1 Pre-processing

The dataset CSV file has been downloaded from the Kaggle website, and the following pre-processing steps were applied:

- Adding suitable headers to the columns: The data was missing headers for the columns so suitable header names were added.
- Dropping null values: text data is crucial in this project so null values must be dropped.
- Checking distribution of classes: calculating the distrbution of classes is extremely important to ensure balance of the data.
- Text cleaning: removal of URLs, mentions, special characters and numbers was applied to prepare the data.
- EDA: a simple exploratory analysis (EDA) was conducted to explore and better visualize the data.
- Emoji decoding: since tweets contain a lot of emojis, emojis were decoded into words to avoid any negative impact on the classification process.
- Tokenization: tokenization was applied for easier machine analysis.
- Stopwords Removal: stopwords don't have a lot of significance when it comes to text classification, thus, it was decided to remove them.

- Lemmatization: words were reduced to their root to facilitate the text classification process for the models.

Moreover, the following text representation methods were used and contrasted against one another to better evaluate models and to identify the best performing combination:

- TF-IDF vectorizer: Term frequency, inverse document frequency converts texttual data into a machine learning algorithm friendly representation of integers or numbers.
- Count Vectorizer: is also a method to transform text data to numerical data to then be used by machine learning models.

Getting the dataset

In [2]:
```
#Validation dataset
val_df = pd.read_csv("twitter_validation.csv", header=None)
#Full dataset for Train-Test
train_df = pd.read_csv("twitter_training.csv", header=None)
```

In [3]:
```
val_df.head()
```

Out[3]:

	0	1	2	3
0	3364	Facebook	Irrelevant	I mentioned on Facebook that I was struggling ...
1	352	Amazon	Neutral	BBC News - Amazon boss Jeff Bezos rejects clai...
2	8312	Microsoft	Negative	@Microsoft Why do I pay for WORD when it funct...
3	4371	CS-GO	Negative	CSGO matchmaking is so full of closet hacking,...
4	4433	Google	Neutral	Now the President is slapping Americans in the...

In [4]:
```
train_df.head()
```

Out[4]:

	0	1	2	3
0	2401	Borderlands	Positive	im getting on borderlands and i will murder yo...
1	2401	Borderlands	Positive	I am coming to the borders and I will kill you...
2	2401	Borderlands	Positive	im getting on borderlands and i will kill you ...
3	2401	Borderlands	Positive	im coming on borderlands and i will murder you...
4	2401	Borderlands	Positive	im getting on borderlands 2 and i will murder ...

Adding column names

In [5]:
```
train_df.columns = ['id','entity','label','text']
train_df.head()
```

Out[5]:

	id	entity	label	text
0	2401	Borderlands	Positive	im getting on borderlands and i will murder yo...
1	2401	Borderlands	Positive	I am coming to the borders and I will kill you...
2	2401	Borderlands	Positive	im getting on borderlands and i will kill you ...
3	2401	Borderlands	Positive	im coming on borderlands and i will murder you...
4	2401	Borderlands	Positive	im getting on borderlands 2 and i will murder ...

In [6]:
```
val_df.columns = ['id', 'entity', 'label', 'text']
val_df.head()
```

Out[6]:

	id	entity	label	text
0	3364	Facebook	Irrelevant	I mentioned on Facebook that I was struggling ...
1	352	Amazon	Neutral	BBC News - Amazon boss Jeff Bezos rejects clai...
2	8312	Microsoft	Negative	@Microsoft Why do I pay for WORD when it funct...
3	4371	CS-GO	Negative	CSGO matchmaking is so full of closet hacking,...
4	4433	Google	Neutral	Now the President is slapping Americans in the...

Explore the structure and contents of the dataset

In [7]:
```
val_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   id      1000 non-null   int64 
 1   entity  1000 non-null   object
 2   label   1000 non-null   object
 3   text    1000 non-null   object
dtypes: int64(1), object(3)
memory usage: 31.4+ KB
```

In [8]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74682 entries, 0 to 74681
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   id      74682 non-null  int64 
 1   entity  74682 non-null  object
 2   label   74682 non-null  object
 3   text    73996 non-null  object
dtypes: int64(1), object(3)
memory usage: 2.3+ MB
```

In [9]:
```
val_df.shape
```

Out[9]:
```
(1000, 4)
```

In [10]:
```
train_df.shape
```

Out[10]:
```
(74682, 4)
```

Removing null values

In [11]:
```
val_df.isnull().sum()
```

Out[11]:
```
id        0
entity    0
label     0
text      0
dtype: int64
```

In [12]:
```
train_df.isnull().sum()
```

Out[12]:
```
id          0
entity      0
label       0
text      686
dtype: int64
```

In [13]:
```
#checking count of null and valid text
null_text = train_df[(train_df.text.isnull())].shape[0]
valid_text = train_df[(train_df.text.notnull())].shape[0]
print(f"Null text count: {null_text}")
print(f"Valid text count: {valid_text}")

#Percentage of null text to valid text
text_percent = null_text/(null_text + valid_text)*100
print(f"Percantage of null text: {text_percent:.{2}f}" + "%")
```

```
Null text count: 686
Valid text count: 73996
Percantage of null text: 0.92%
```

There are 686 null textx in the training data, since the text is cruicial in this project we will drop the null text.

In [14]:
```
# drop samples with nans
train_df.dropna(inplace=True, axis=0)
```

In [15]:
```
train_df.isnull().sum()
```

Out[15]:
```
id        0
entity    0
label     0
text      0
dtype: int64
```

Checking sentiments distribution

In [16]:
```
# Check the distribution of Emotion in training set
train_df['label'].value_counts()
```

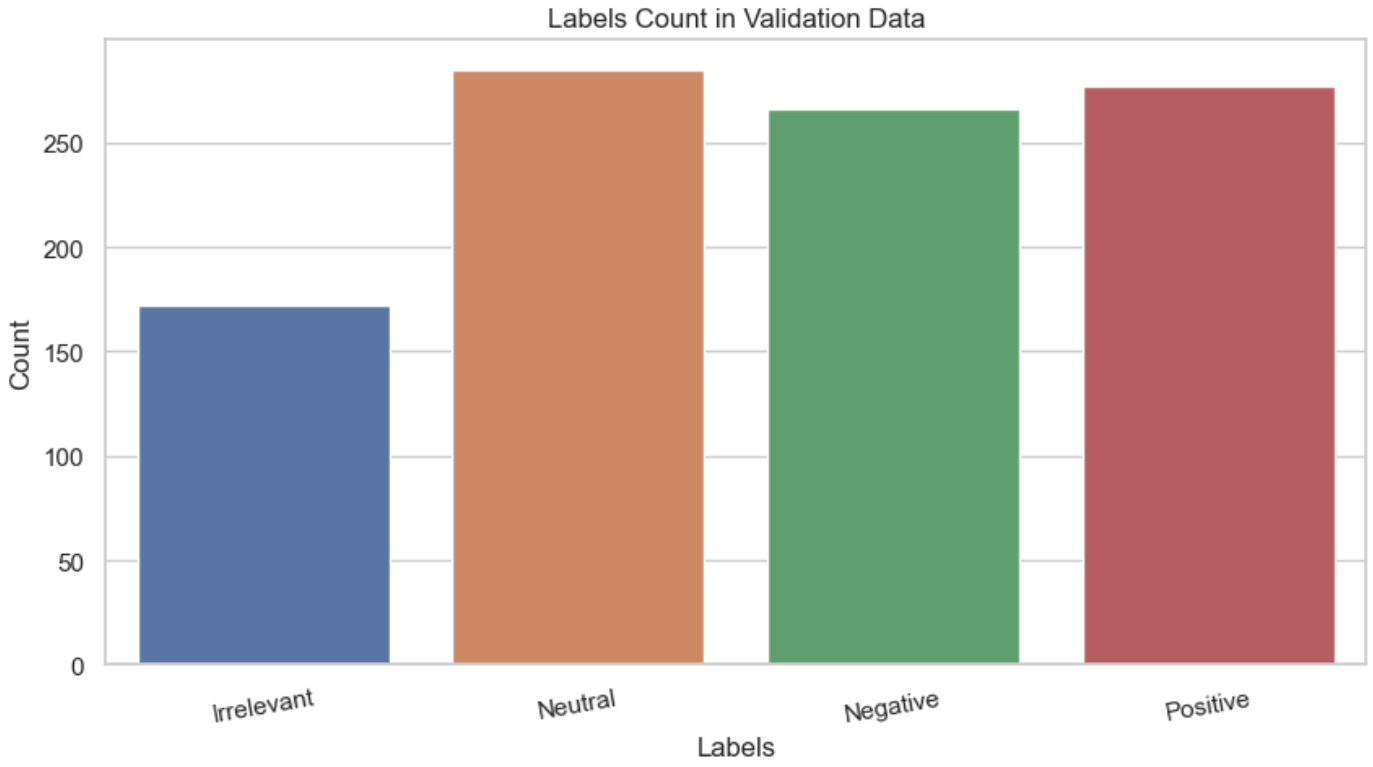
Out[16]:
```
label
Negative      22358
Positive      20655
Neutral       18108
Irrelevant    12875
Name: count, dtype: int64
```

In [17]:
```
plt.figure(figsize=(10, 5))
sns.countplot(data=train_df, x='label', palette='deep')
# set labels and title
plt.xlabel('Labels')
plt.ylabel('Count')
plt.title('Labels Count in Training Data')

# Rotate x-axis labels for better readability
plt.xticks(rotation=10)
# display the plot
plt.show()
```

In [18]:
```
# Check the distribution of Emotion in validation set
val_df['label'].value_counts()
```

Out[18]:
```
label
Neutral       285
Positive      277
Negative      266
Irrelevant    172
Name: count, dtype: int64
```

In [19]:
```
plt.figure(figsize=(10, 5))
sns.countplot(data=val_df, x='label', palette='deep')
# set labels and title
plt.xlabel('Labels')
plt.ylabel('Count')
plt.title('Labels Count in Validation Data')

# Rotate x-axis labels for better readability
plt.xticks(rotation=10)
# display the plot
plt.show()
```



Target categories are almost of equal size, so resampling is not required.

Text cleaning

In [20]:
```
def clean_text(text):
    # Remove URLs
    text = re.sub(r'http\S+', '', text)

    # Remove mentions
    text = re.sub(r'@\S+', '', text)

    # Remove special characters and numbers
```

```
    text = re.sub(r'[^A-Za-z\s]', '', text)

    return text

# Apply the clean_text function to the 'text' column
train_df['clean_text'] = train_df['text'].apply(clean_text)
val_df['clean_text'] = val_df['text'].apply(clean_text)
```

In [21]:
```
# converting to lowercase string
train_df["clean_text"] = train_df["clean_text"].str.lower()
val_df["clean_text"] = val_df["clean_text"].str.lower()
```

In [22]:
```
# comparison between text and clean_text columns
train_df.head()
```

Out[22]:

	id	entity	label	text	clean_text
0	2401	Borderlands	Positive	im getting on borderlands and i will murder yo...	im getting on borderlands and i will murder yo...
1	2401	Borderlands	Positive	I am coming to the borders and I will kill you...	i am coming to the borders and i will kill you...
2	2401	Borderlands	Positive	im getting on borderlands and i will kill you ...	im getting on borderlands and i will kill you all
3	2401	Borderlands	Positive	im coming on borderlands and i will murder you...	im coming on borderlands and i will murder you...
4	2401	Borderlands	Positive	im getting on borderlands 2 and i will murder ...	im getting on borderlands and i will murder y...

In [23]:
```
val_df.head()
```

Out[23]:

	id	entity	label	text	clean_text
0	3364	Facebook	Irrelevant	I mentioned on Facebook that I was struggling ...	i mentioned on facebook that i was struggling ...
1	352	Amazon	Neutral	BBC News - Amazon boss Jeff Bezos rejects clai...	bbc news amazon boss jeff bezos rejects claim...
2	8312	Microsoft	Negative	@Microsoft Why do I pay for WORD when it funct...	why do i pay for word when it functions so po...
3	4371	CS-GO	Negative	CSGO matchmaking is so full of closet hacking,...	csgo matchmaking is so full of closet hacking ...
4	4433	Google	Neutral	Now the President is slapping Americans in the...	now the president is slapping americans in the...

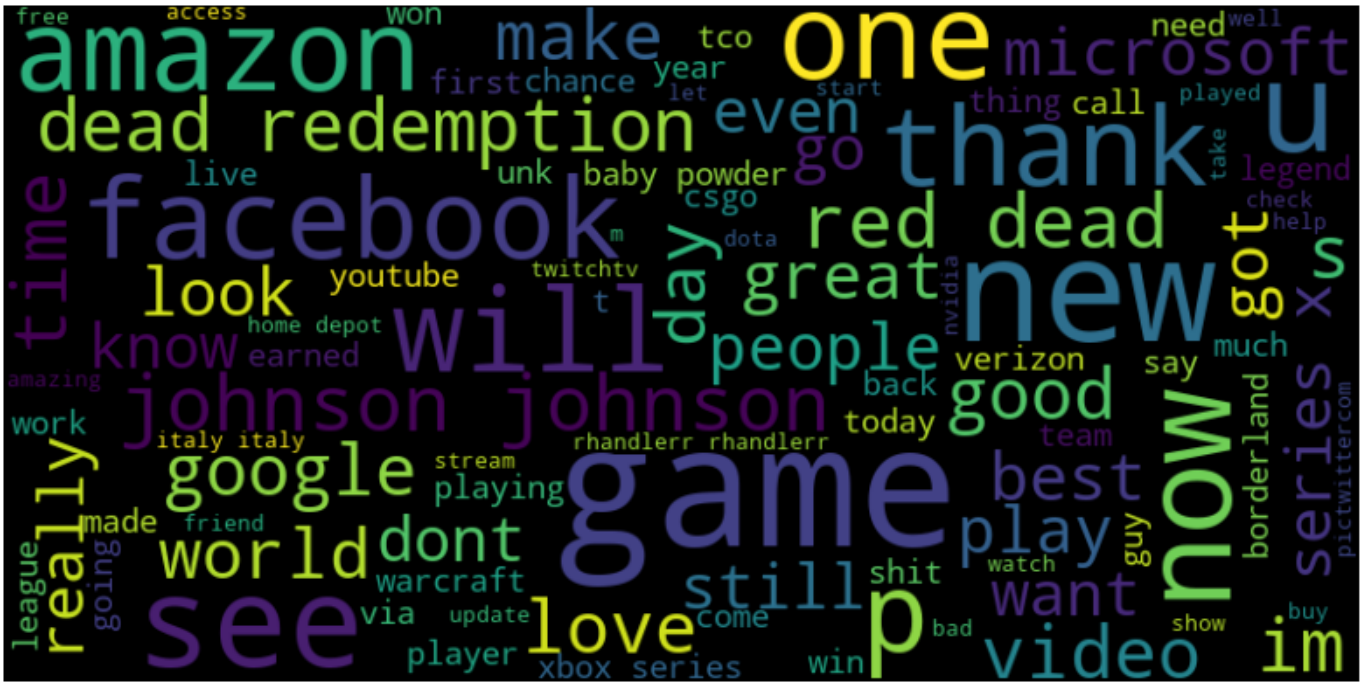
EDA: WordClouds

In [24]:
```
wordcloud_text = ''.join(train_df[train_df['label']=='Positive']['clean_text'])
# Create the word cloud
wordcloud = WordCloud(max_font_size=100, max_words=100, width=800, height=400,
```

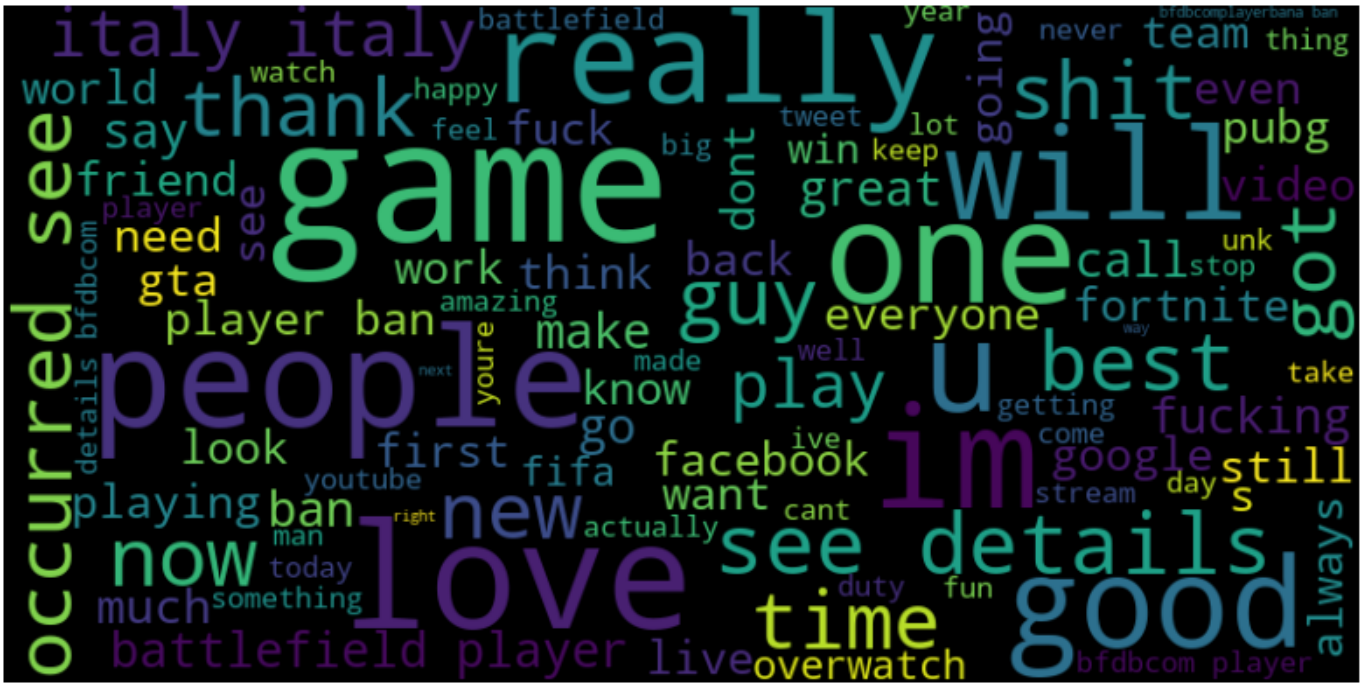
```
background_color='black').generate(wordcloud_text)

# Display the word cloud
plt.figure(figsize=(10, 10))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

In [25]:
```
wordcloud_text = ''.join(train_df[train_df['label']=='Negative']['clean_text'])
# Create the word cloud
wordcloud = WordCloud(max_font_size=100, max_words=100, width=800, height=400,
background_color='black').generate(wordcloud_text)

# Display the word cloud
plt.figure(figsize=(10, 10))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

In [26]:
```
wordcloud_text = ''.join(train_df[train_df['label']=='Neutral']['clean_text'])
# Create the word cloud
wordcloud = WordCloud(max_font_size=100, max_words=100, width=800, height=400,
background_color='black').generate(wordcloud_text)

# Display the word cloud
plt.figure(figsize=(10, 10))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



In [27]:
```
wordcloud_text = ''.join(train_df[train_df['label']=='Irrelevant']['clean_text'])
# Create the word cloud
wordcloud = WordCloud(max_font_size=100, max_words=100, width=800, height=400,
background_color='black').generate(wordcloud_text)

# Display the word cloud
plt.figure(figsize=(10, 10))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



Emoji Decoding

In [28]:
```
def emoji_to_word(text):
    return emoji.demojize(text)

# Apply emoji_to_word function to the 'clean_text' column
train_df['clean_text'] = train_df['clean_text'].apply(emoji_to_word)
val_df['clean_text'] = val_df['clean_text'].apply(emoji_to_word)
```

In [29]:
```
# Tokenize the cleaned text
train_df['tokens'] = train_df['clean_text'].apply(word_tokenize)
val_df['tokens'] = val_df['clean_text'].apply(word_tokenize)
```

In [30]:
```
train_df.head()
```

Out[30]:

	id	entity	label	text	clean_text	tokens
0	2401	Borderlands	Positive	im getting on borderlands and i will murder yo...	im getting on borderlands and i will murder yo...	[im, getting, on, borderlands, and, i, will, m...
1	2401	Borderlands	Positive	I am coming to the borders and I will kill you...	i am coming to the borders and i will kill you...	[i, am, coming, to, the, borders, and, i, will...
2	2401	Borderlands	Positive	im getting on borderlands and i will kill you ...	im getting on borderlands and i will kill you all	[im, getting, on, borderlands, and, i, will, k...
3	2401	Borderlands	Positive	im coming on borderlands and i will murder you...	im coming on borderlands and i will murder you...	[im, coming, on, borderlands, and, i, will, mu...
4	2401	Borderlands	Positive	im getting on borderlands 2 and i will murder ...	im getting on borderlands and i will murder y...	[im, getting, on, borderlands, and, i, will, m...

In [31]:
```
val_df.head()
```

Out[31]:

	id	entity	label	text	clean_text	tokens
0	3364	Facebook	Irrelevant	I mentioned on Facebook that I was struggling ...	i mentioned on facebook that i was struggling ...	[i, mentioned, on, facebook, that, i, was, str...
1	352	Amazon	Neutral	BBC News - Amazon boss Jeff Bezos rejects clai...	bbc news amazon boss jeff bezos rejects claim...	[bbc, news, amazon, boss, jeff, bezos, rejects...
2	8312	Microsoft	Negative	@Microsoft Why do I pay for WORD when it funct...	why do i pay for word when it functions so po...	[why, do, i, pay, for, word, when, it, functio...
3	4371	CS-GO	Negative	CSGO matchmaking is so full of closet hacking,...	csgo matchmaking is so full of closet hacking ...	[csgo, matchmaking, is, so, full, of, closet, ...
4	4433	Google	Neutral	Now the President is slapping Americans in the...	now the president is slapping americans in the...	[now, the, president, is, slapping, americans,...

In [32]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 73996 entries, 0 to 74681
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   id          73996 non-null  int64 
 1   entity      73996 non-null  object
 2   label       73996 non-null  object
 3   text        73996 non-null  object
 4   clean_text  73996 non-null  object
 5   tokens      73996 non-null  object
dtypes: int64(1), object(5)
memory usage: 4.0+ MB
```

Removing Stopwords and Lemmatization

In [33]:
```
# Initialize the lemmatizer
lemmatizer = WordNetLemmatizer()

# Define a function to remove stopwords, lemmatize and join tokens back into text
def process_text(tokens):
    stopwords_nltk = stopwords.words('english')
    stop_words = set(stopwords_nltk)

    # Lemmatize each token
    lemmatized_tokens = [lemmatizer.lemmatize(token, pos='v') for token in tokens]
    # Remove stopwords
    filtered_tokens = [token for token in lemmatized_tokens if token.lower() not in stop_words]

    return ' '.join(filtered_tokens)

# Apply the process_text function to the 'tokens' column
train_df['processed_text'] = train_df['tokens'].apply(process_text)
val_df['processed_text'] = val_df['tokens'].apply(process_text)
```

In [34]:
```
train_df.head()
```

Out[34]:

	id	entity	label	text	clean_text	tokens	processed_text
0	2401	Borderlands	Positive	im getting on borderlands and i will murder yo...	im getting on borderlands and i will murder yo...	[im, getting, on, borderlands, and, i, will, m...	im get borderlands murder
1	2401	Borderlands	Positive	I am coming to the borders and I will kill you...	i am coming to the borders and i will kill you...	[i, am, coming, to, the, borders, and, i, will...	come border kill
2	2401	Borderlands	Positive	im getting on borderlands and i will kill you ...	im getting on borderlands and i will kill you all	[im, getting, on, borderlands, and, i, will, k...	im get borderlands kill
3	2401	Borderlands	Positive	im coming on borderlands and i will murder you...	im coming on borderlands and i will murder you...	[im, coming, on, borderlands, and, i, will, mu...	im come borderlands murder
4	2401	Borderlands	Positive	im getting on borderlands 2 and i will murder ...	im getting on borderlands and i will murder y...	[im, getting, on, borderlands, and, i, will, m...	im get borderlands murder

In [35]:
```
val_df.head()
```

Out[35]:

	id	entity	label	text	clean_text	tokens	processed_text
0	3364	Facebook	Irrelevant	I mentioned on Facebook that I was struggling ...	i mentioned on facebook that i was struggling ...	[i, mentioned, on, facebook, that, i, was, str...	mention facebook struggle motivation go run da...
1	352	Amazon	Neutral	BBC News - Amazon boss Jeff Bezos rejects clai...	bbc news amazon boss jeff bezos rejects claim...	[bbc, news, amazon, boss, jeff, bezos, rejects...	bbc news amazon boss jeff bezos reject claim c...
2	8312	Microsoft	Negative	@Microsoft Why do I pay for WORD when it funct...	why do i pay for word when it functions so po...	[why, do, i, pay, for, word, when, it, functio...	pay word function poorly chromebook
3	4371	CS-GO	Negative	CSGO matchmaking is so full of closet hacking,...	csgo matchmaking is so full of closet hacking ...	[csgo, matchmaking, is, so, full, of, closet, ...	csgo matchmaking full closet hack truly awful ...
4	4433	Google	Neutral	Now the President is slapping Americans in the...	now the president is slapping americans in the...	[now, the, president, is, slapping, americans,...	president slap americans face really commit un...

In [36]:
```
# Drop unnecessary columns
train_df.drop(['clean_text', 'tokens'], axis=1, inplace=True)
val_df.drop(['clean_text', 'tokens'], axis=1, inplace=True)
```

In [37]:
```
train_df.head()
```

Out[37]:

	id	entity	label	text	processed_text
0	2401	Borderlands	Positive	im getting on borderlands and i will murder yo...	im get borderlands murder
1	2401	Borderlands	Positive	I am coming to the borders and I will kill you...	come border kill
2	2401	Borderlands	Positive	im getting on borderlands and i will kill you ...	im get borderlands kill
3	2401	Borderlands	Positive	im coming on borderlands and i will murder you...	im come borderlands murder
4	2401	Borderlands	Positive	im getting on borderlands 2 and i will murder ...	im get borderlands murder

In [38]:
```
val_df.head()
```

Out[38]:

	id	entity	label	text	processed_text
0	3364	Facebook	Irrelevant	I mentioned on Facebook that I was struggling ...	mention facebook struggle motivation go run da...
1	352	Amazon	Neutral	BBC News - Amazon boss Jeff Bezos rejects clai...	bbc news amazon boss jeff bezos reject claim c...
2	8312	Microsoft	Negative	@Microsoft Why do I pay for WORD when it funct...	pay word function poorly chromebook
3	4371	CS-GO	Negative	CSGO matchmaking is so full of closet hacking,...	csgo matchmaking full closet hack truly awful ...
4	4433	Google	Neutral	Now the President is slapping Americans in the...	president slap americans face really commit un...

Splitting Train data

In [39]:
```
# encoding target column
le_model = LabelEncoder()
train_df['label'] = le_model.fit_transform(train_df['label'])
```

In [40]:
```
train_df.head(5)
```

Out[40]:

	id	entity	label	text	processed_text
0	2401	Borderlands	3	im getting on borderlands and i will murder yo...	im get borderlands murder
1	2401	Borderlands	3	I am coming to the borders and I will kill you...	come border kill
2	2401	Borderlands	3	im getting on borderlands and i will kill you ...	im get borderlands kill
3	2401	Borderlands	3	im coming on borderlands and i will murder you...	im come borderlands murder
4	2401	Borderlands	3	im getting on borderlands 2 and i will murder ...	im get borderlands murder

In [41]:
```
print(train_df['label'].unique())
```

```
[3 2 1 0]
```

In [42]:
```
# Split data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(train_df['processed_text'],
train_df['label'], test_size=0.1, random_state=42, stratify=train_df['label'])
```

In [43]:
```
print("X_train shape:", X_train.shape)
```

```
X_train shape: (66596,)
```

In [44]:
```
print("X_val shape:", X_val.shape)
```

```
X_val shape: (7400,)
```

2.2 Baseline Performance

Logistic Regression with Count Vectorizer/TF-IDF

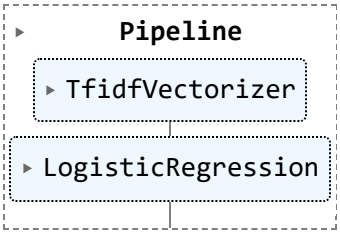
The logistic regression model's results will be used a baseline. Logistic Regression is a one of the most widely used algorithms for classification and it performs really well on linearly seperable classes. The Logistic Regression model has been used with both TF-IDF and count vectorizer to assess its performance with different vectorization methods. For evaluation, accuracy and a full classification report were used.
Implementation of Logistic Regression with TF-IDF achieved an accuracy of 76%, while its implementation with Count vectorizer achieved an accuracy of 80%, the full classification report is available below. The latter results will serve as a benchmark for our project.
This model's simplicity makes it an appropriate baseline for our problem, and its performance results are a great reference point for more complex models that we will implement later on.

Logistic Regression with TF-IDF vectorizer

In [94]:
```
lr_clf1 = Pipeline([
    ('vectorizer_tri_grams', TfidfVectorizer()),
    ('logistic_regression', (LogisticRegression(C=1, max_iter=1000)))
])
```

In [95]:
```
# Model training
lr_clf1.fit(X_train, y_train)
```

Out[95]:



In [96]:
```
# Get prediction
y_pred = lr_clf1.predict(X_val)
```

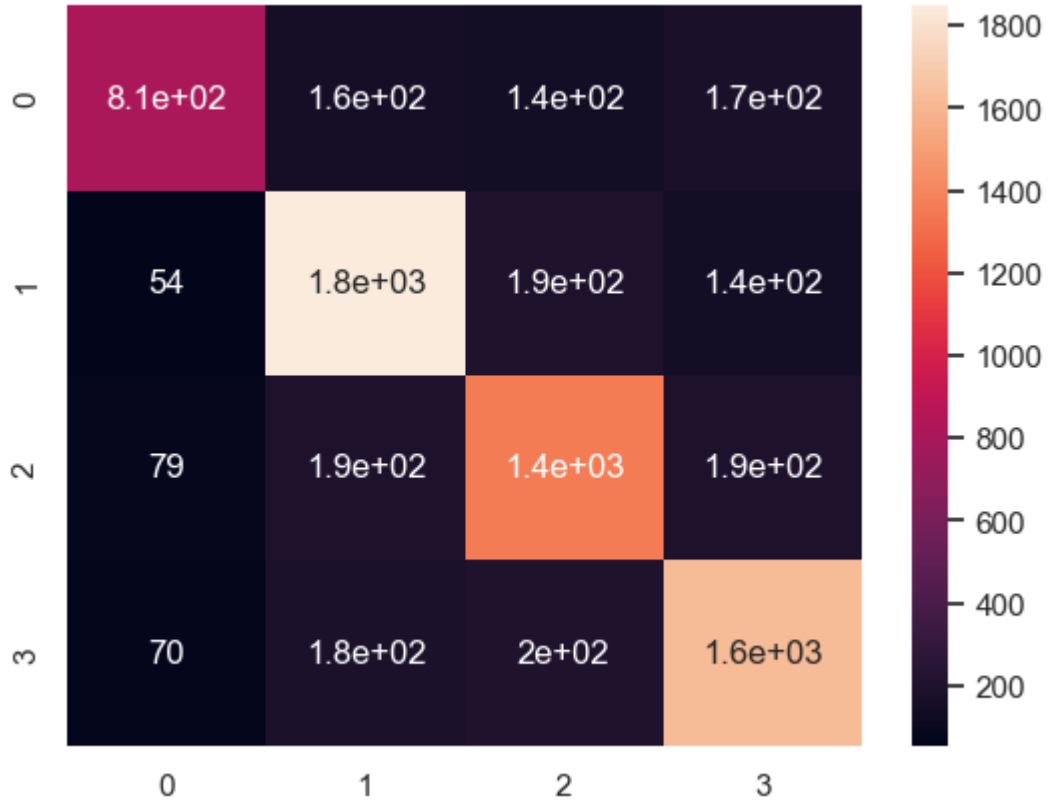
In [97]:
```
# Print score
print(accuracy_score(y_val, y_pred))
```

```
0.760945945945946
```

In [98]:
```
# Print classification report
print(classification_report(y_val, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.63      0.71      1287
           1       0.78      0.83      0.80      2236
           2       0.72      0.75      0.73      1811
           3       0.76      0.78      0.77      2066

    accuracy                           0.76      7400
   macro avg       0.76      0.75      0.75      7400
weighted avg       0.76      0.76      0.76      7400
```

In [99]:
```
sns.heatmap(confusion_matrix(y_val, y_pred), annot=True)
plt.show()
```

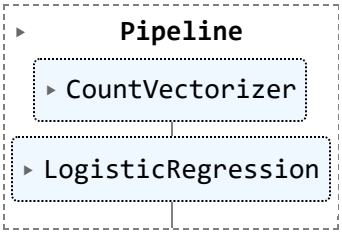


Logistic Regression with Count vectorizer

In [100…
```
lr_clf2 = Pipeline([
    ('vectorizer_count', CountVectorizer()),
    ('logistic_regression', (LogisticRegression(C=1, max_iter=1000)))
])
```

In [101…
```
# Model training
lr_clf2.fit(X_train, y_train)
```

Out[101]:



In [102…
```
# Get prediction
y_pred = lr_clf2.predict(X_val)
```

In [103…
```
# Print score
print(accuracy_score(y_val, y_pred))
```

```
0.7994594594594595
```

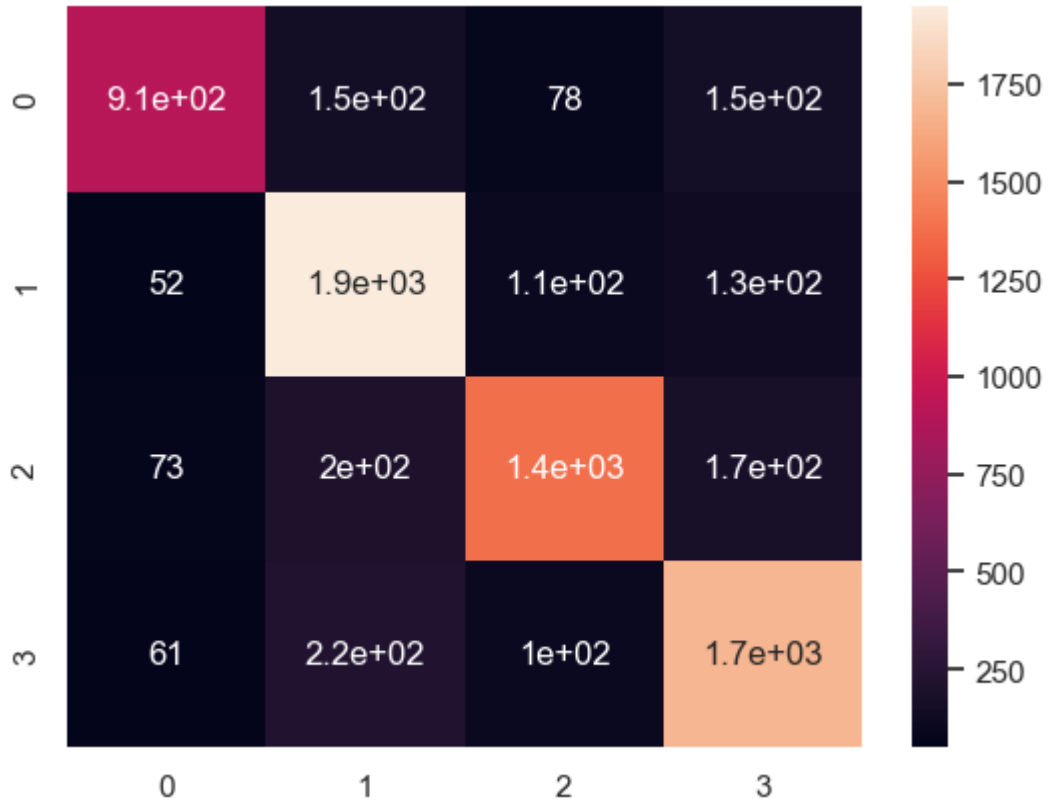
In [105…

```
# Print classification report
print(classification_report(y_val, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.71      0.76      1287
           1       0.78      0.87      0.82      2236
           2       0.83      0.76      0.79      1811
           3       0.79      0.82      0.80      2066

    accuracy                           0.80      7400
   macro avg       0.81      0.79      0.79      7400
weighted avg       0.80      0.80      0.80      7400
```

In [106…

```
sns.heatmap(confusion_matrix(y_val, y_pred), annot=True)
plt.show()
```



As seen above, Logistic Regression has performed better with Count Vectorizer reaching an accuracy of 80%, and the confusion matrices look diagonal with all classes predicted almost equally.

2.3 Classification Approach

2.3.1 Testing different models: SVM, Random Forest, KNN

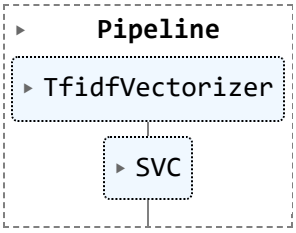
Different models that are well-known for text-classification will be implemented with different vectorization techniques (TF-IDF and Count Vectorizer) to identify the best performing one and improve it further.

SVM with TF-IDF

In [107…]
```
svm_clf1 = Pipeline([
    ('vectorizer_tri_grams', TfidfVectorizer()),
    ('svm', (SVC(kernel='linear')))
])
```

In [108…]
```
svm_clf1.fit(X_train, y_train)
```

Out[108]:



In [109…]
```
y_pred = svm_clf1.predict(X_val)
```

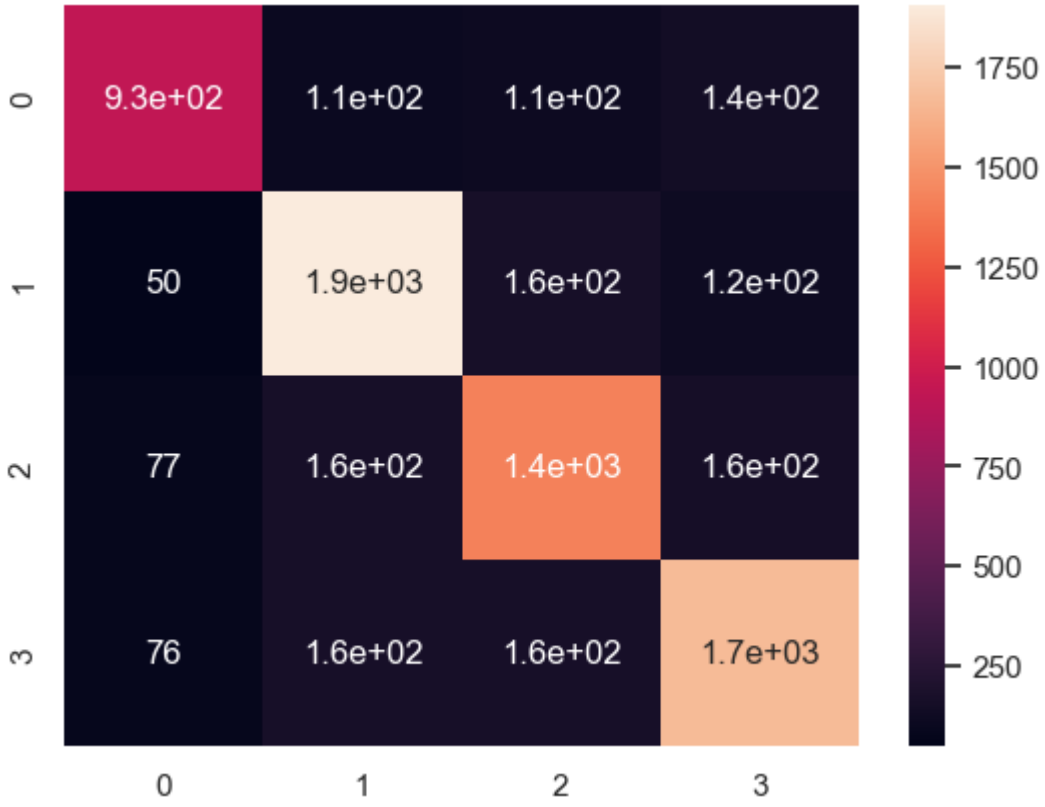
In [110…]
```
# Print accuracy score
print(accuracy_score(y_val, y_pred))
```

```
0.8002702702702703
```

In [111…]
```
# Print classification report
print(classification_report(y_val, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.72      0.77      1287
           1       0.82      0.85      0.83      2236
           2       0.77      0.78      0.77      1811
           3       0.80      0.81      0.80      2066

    accuracy                           0.80      7400
   macro avg       0.80      0.79      0.80      7400
weighted avg       0.80      0.80      0.80      7400
```

In [112…]
```
sns.heatmap(confusion_matrix(y_val, y_pred), annot=True)
plt.show()
```

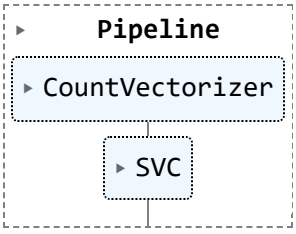


SVM with count vectorizer

In [113…
```
svm_clf2 = Pipeline([
    ('vectorizer_count', CountVectorizer()),
    ('svm', (SVC(kernel='linear')))
])
```

In [114…
```
svm_clf2.fit(X_train, y_train)
```

Out[114]:



In [115…
```
y_pred = svm_clf2.predict(X_val)
```

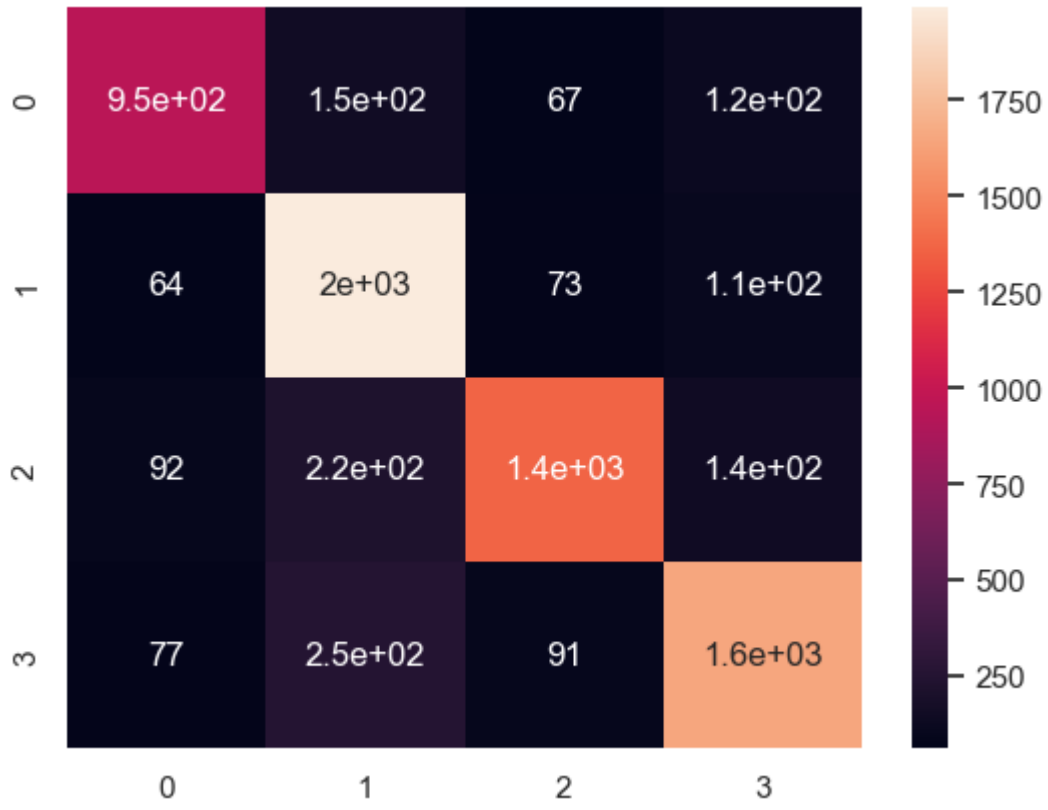
In [116…
```
# Print accuracy score
print(accuracy_score(y_val, y_pred))
```

```
0.8041891891891891
```

In [117…
```
# Print classification report
print(classification_report(y_val, y_pred))
```

	precision	recall	f1-score	support
0	0.80	0.74	0.77	1287
1	0.76	0.89	0.82	2236
2	0.85	0.75	0.80	1811
3	0.82	0.80	0.81	2066
accuracy			0.80	7400
macro avg	0.81	0.79	0.80	7400
weighted avg	0.81	0.80	0.80	7400

In [118…
```
sns.heatmap(confusion_matrix(y_val, y_pred), annot=True)
plt.show()
```



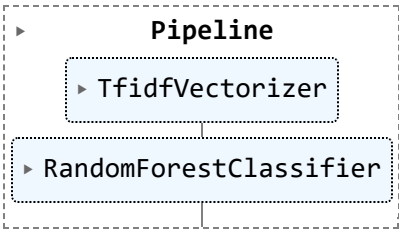
As seen above, SVM has performed better with Count Vectorizer reaching an accuracy of ~80%, and the confusion matrices look diagonal with all classes predicted almost equally.

Random forest with TF-IDF

In [119…
```
rf_clf1 = Pipeline([
    ('vectorizer_tri_grams', TfidfVectorizer()),
    ('random_forest', (RandomForestClassifier()))
])
```

In [120…
```
rf_clf1.fit(X_train, y_train)
```

Out[120]:



In [121…
```
y_pred = rf_clf1.predict(X_val)
```

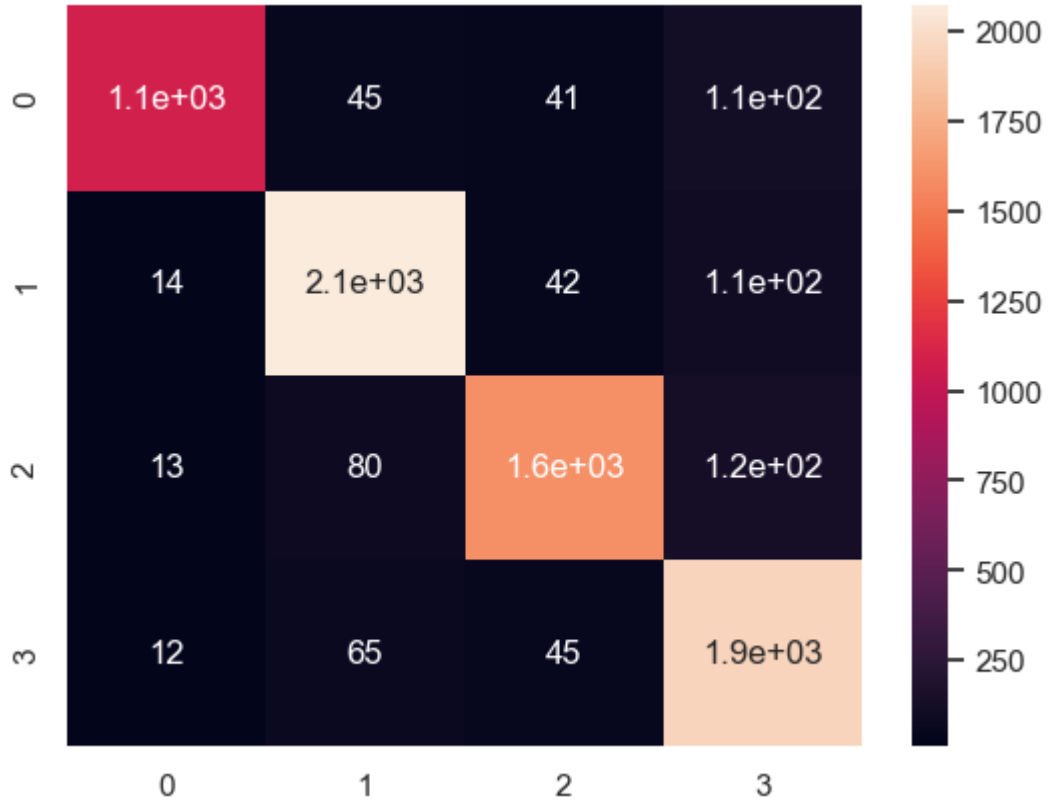
In [122…
```
# Print accuracy score
print(accuracy_score(y_val, y_pred))
```

```
0.9051351351351351
```

In [123…
```
# Print classification report
print(classification_report(y_val, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.85      0.90      1287
           1       0.92      0.93      0.92      2236
           2       0.93      0.88      0.90      1811
           3       0.85      0.94      0.89      2066

    accuracy                           0.91      7400
   macro avg       0.91      0.90      0.90      7400
weighted avg       0.91      0.91      0.91      7400
```

In [124…
```
sns.heatmap(confusion_matrix(y_val, y_pred), annot=True)
plt.show()
```

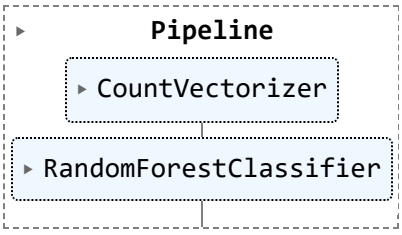


Random Forest with Count vectorizer

In [125…
```
rf_clf2 = Pipeline([
    ('vectorizer_count', CountVectorizer()),
    ('random_forest', (RandomForestClassifier()))
])
```

In [126…
```
rf_clf2.fit(X_train, y_train)
```

Out[126]:



In [127…
```
y_pred = rf_clf2.predict(X_val)
```

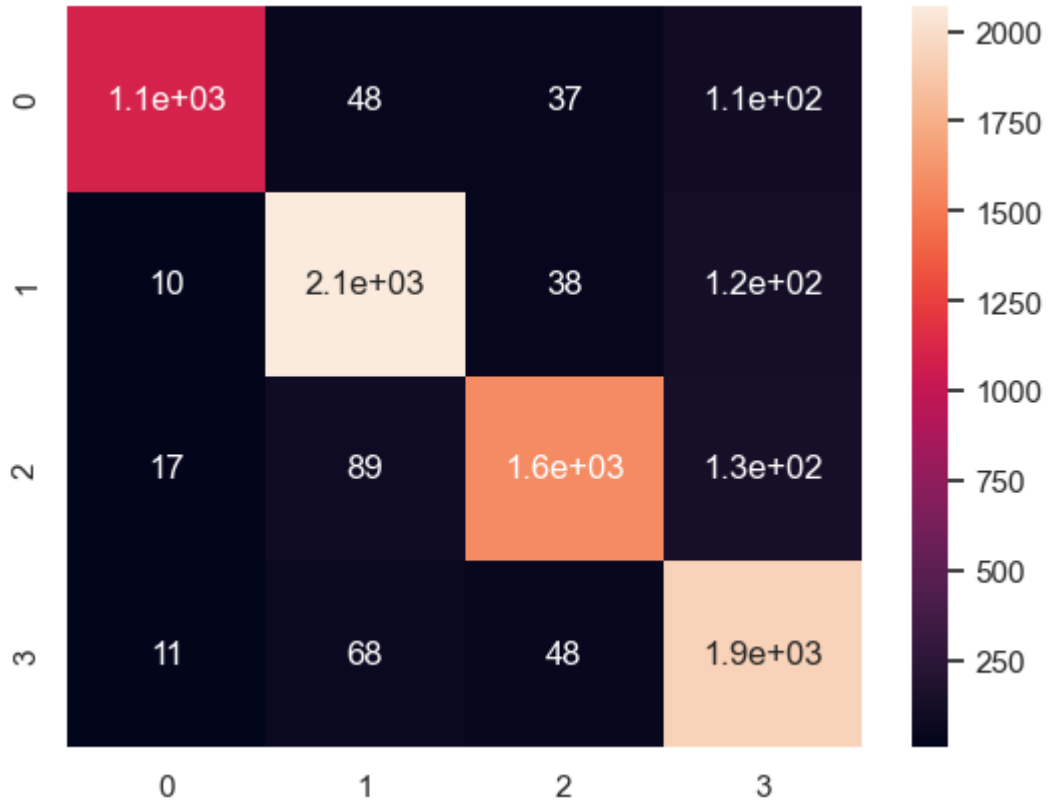
In [128…
```
# Print accuracy score
print(accuracy_score(y_val, y_pred))
```

```
0.9021621621621622
```

In [129…
```
# Print classification report
print(classification_report(y_val, y_pred))
```

	precision	recall	f1-score	support
0	0.97	0.85	0.91	1287
1	0.91	0.93	0.92	2236
2	0.93	0.87	0.90	1811
3	0.84	0.94	0.89	2066
accuracy			0.90	7400
macro avg	0.91	0.90	0.90	7400
weighted avg	0.91	0.90	0.90	7400

In [130…
```
sns.heatmap(confusion_matrix(y_val, y_pred), annot=True)
plt.show()
```



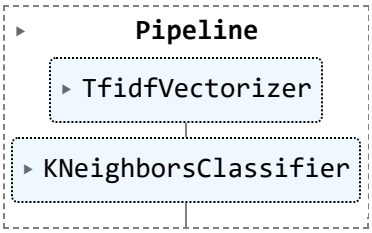
As seen above, Random Forest has performed better with TF-IDF Vectorizer reaching an accuracy of ~91%, and the confusion matrices look diagonal with all classes predicted almost equally.

KNN with TF-IDF

In [131…
```
knn_clf1 = Pipeline([
    ('vectorizer_tri_grams', TfidfVectorizer()),
    ('knn', KNeighborsClassifier(n_neighbors=5))
])
```

In [132…
```
knn_clf1.fit(X_train, y_train)
```

Out[132]:



In [133…
```
y_pred = knn_clf1.predict(X_val)
```

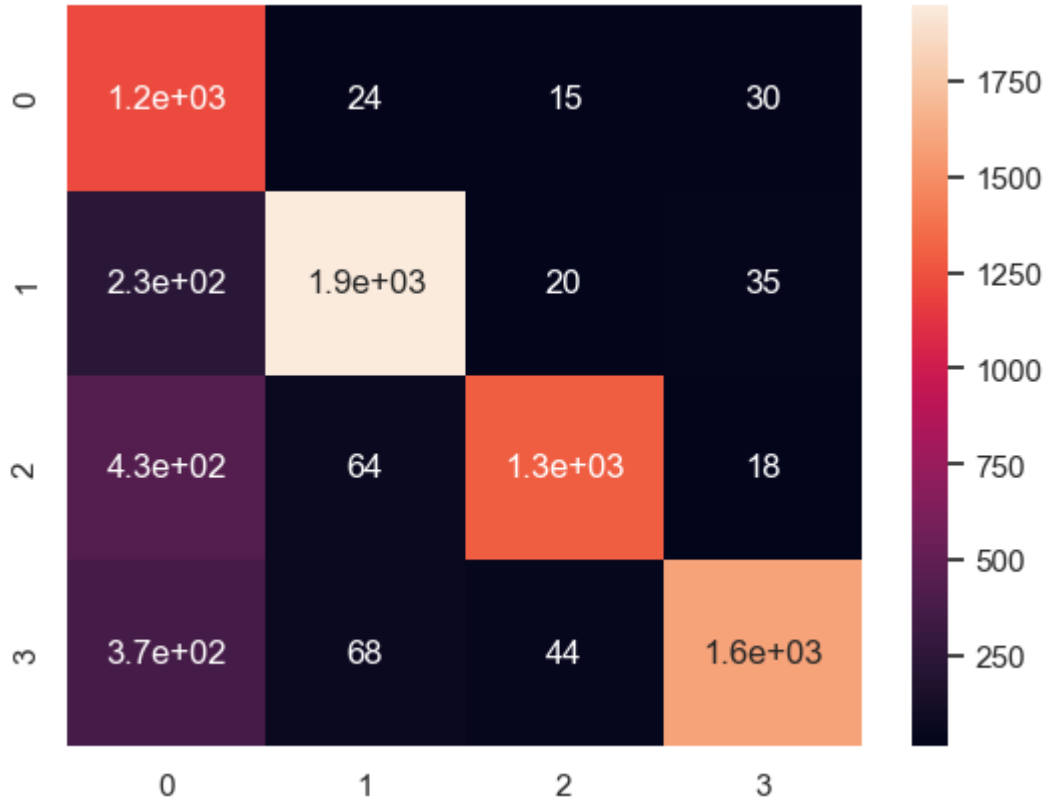
In [134…
```
# Print accuracy score
print(accuracy_score(y_val, y_pred))
```

```
0.8175675675675675
```

In [135…
```
# Print classification report
print(classification_report(y_val, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.54      0.95      0.69      1287
           1       0.93      0.87      0.90      2236
           2       0.94      0.72      0.81      1811
           3       0.95      0.77      0.85      2066

    accuracy                           0.82      7400
   macro avg       0.84      0.83      0.81      7400
weighted avg       0.87      0.82      0.83      7400
```

In [136…
```
sns.heatmap(confusion_matrix(y_val, y_pred), annot=True)
plt.show()
```

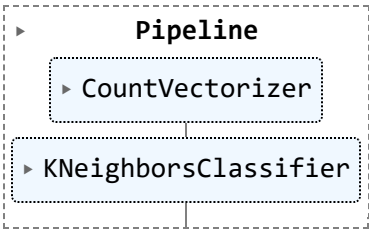


KNN with Count vectorizer

In [137…
```
# Create a pipeline for Gradient Boosting
knn_clf2 = Pipeline([
    ('vectorizer_count', CountVectorizer()),
    ('knn', KNeighborsClassifier(n_neighbors=5))
])
```

In [138…
```
knn_clf2.fit(X_train, y_train)
```

Out[138]:



In [139…
```
y_pred = knn_clf2.predict(X_val)
```

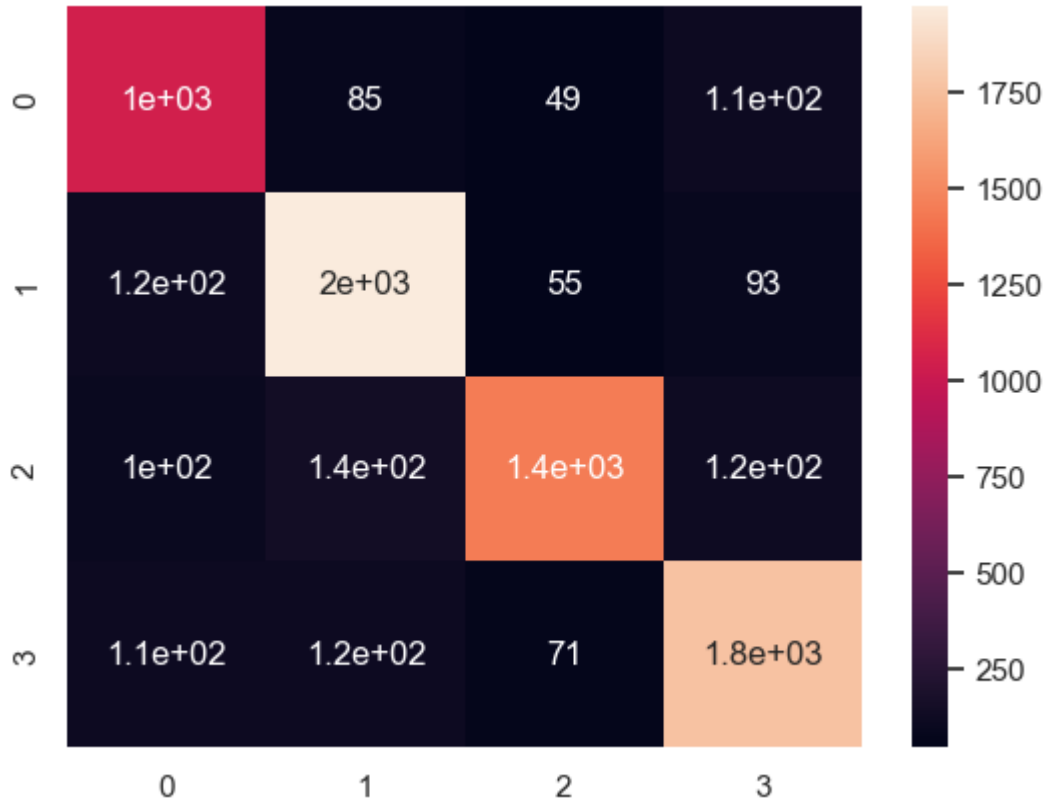
In [140…
```
# Print accuracy score
print(accuracy_score(y_val, y_pred))
```

```
0.8412162162162162
```

In [141…
```
# Print classification report
print(classification_report(y_val, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.76      0.81      0.78      1287
           1       0.85      0.88      0.87      2236
           2       0.89      0.80      0.84      1811
           3       0.84      0.86      0.85      2066

    accuracy                           0.84      7400
   macro avg       0.84      0.84      0.84      7400
weighted avg       0.84      0.84      0.84      7400
```

In [142…]
```
sns.heatmap(confusion_matrix(y_val, y_pred), annot=True)
plt.show()
```



As seen above, KNN has performed better with Count Vectorizer reaching an accuracy of 84%, and the confusion matrices look diagonal with all classes predicted almost equally.

2.3.2 Using RandomizedSearchCV to improve model

RandomizedSearchCV is hyperparameter tuning technique that moves within the given hyperparameter grid in a random fashion to find the best set of hyperparameters. We will be using the latter method on the best over-all performing classification combination: Random Forest with TF-IDF vectorizer, to further improve its results.

In [64]:
```
# TF-IDF vectorization
vectorizer = TfidfVectorizer()
```

```
X_train_tfidf = vectorizer.fit_transform(X_train)
X_val_tfidf = vectorizer.transform(X_val)
```

In [65]:
```
# Define your Random Forest classifier
rf_clf = RandomForestClassifier(random_state=42)
```

In [76]:
```
# Define hyperparameter grid for RandomizedSearchCV
param_dist = {
    'n_estimators': [150, 175, 200, 225, 250],
    'max_features': ['sqrt'],
    'max_depth': [None],
    'min_samples_split': [3, 4, 5, 6, 7],
    'min_samples_leaf': [1, 2, 3],
    'criterion': ['entropy'],
    'bootstrap': [True]
}
```

In [77]:
```
# Number of random combinations to try
n_iter_search = 30
```

In [78]:
```
# Create a StratifiedKFold for cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

In [79]:
```
# Create a RandomizedSearchCV object
random_search = RandomizedSearchCV(
    estimator=rf_clf,
    param_distributions=param_dist,
    scoring='accuracy',
    cv=cv,
    n_iter=n_iter_search,
    random_state=42,
    error_score='raise',
    n_jobs=-1
)
```

In [80]:
```
# Start the timer
start_time = time.time()

# Fit RandomizedSearchCV on the vectorized data
random_search.fit(X_train_tfidf, y_train)

# Stop the timer
end_time = time.time()

# Calculate and print the elapsed time
elapsed_time = end_time - start_time
print(f"Training took {elapsed_time} seconds.")
```

```
Training took 19430.886248588562 seconds.
```

In [81]:
```
# Get the best parameters and model
best_params = random_search.best_params_
best_rf_model = random_search.best_estimator_

# Print the best parameters
print("Best Parameters:", best_params)
```

```
Best Parameters: {'n_estimators': 250, 'min_samples_split': 3, 'min_samples_leaf': 1, 'max_featu
res': 'sqrt', 'max_depth': None, 'criterion': 'entropy', 'bootstrap': True}
```

In [82]:
```
# Transform validation set using the same vectorizer
X_val_tfidf = vectorizer.transform(X_val)
```

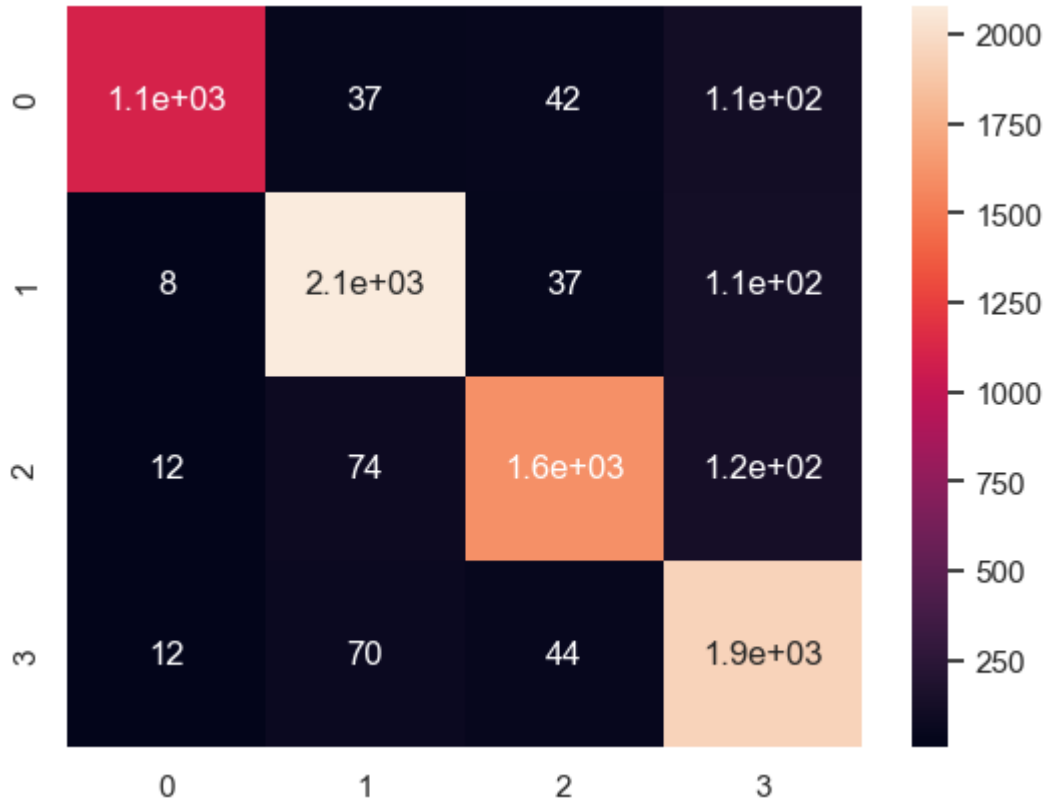
In [83]:
```
# Evaluate the best model on your validation data
y_pred_val = best_rf_model.predict(X_val_tfidf)
accuracy_val = accuracy_score(y_val, y_pred_val)
print("Validation Accuracy:", accuracy_val)
```

```
Validation Accuracy: 0.9081081081081082
```

In [84]:
```
# Print classification report
print(classification_report(y_val, y_pred_val))
```

```
              precision    recall  f1-score   support

           0       0.97      0.85      0.91      1287
           1       0.92      0.93      0.92      2236
           2       0.93      0.89      0.91      1811
           3       0.85      0.94      0.89      2066

    accuracy                           0.91      7400
   macro avg       0.92      0.90      0.91      7400
weighted avg       0.91      0.91      0.91      7400
```

In [85]:
```
sns.heatmap(confusion_matrix(y_val, y_pred_val), annot=True)
plt.show()
```



As we can see, using randomizedSearch on Random Forest which has reached an accuracy of 90.5% before, has now increased to 90.8%. Thus, the hyperparameter tuning process has had a positive impact on the model's performance, especially considering that models that reach higher accuracy levels make achieving substantial improvements more challenging.

2.3.3 Cross-Validation on Random Forest

The final step of model enhancement and optimization is to perform cross-validation on the improved Random Forest to assess its performance and ensure stability and consistency.

In [143…
```
# Perform cross-validation on the training set and get the accuracy scores
cv_scores = cross_val_score(best_rf_model, X_train_tfidf, y_train, cv=cv, scoring='accuracy')
```

In [144…
```
# Print the cross-validation scores
print("Cross-validation scores:", cv_scores)
```

```
Cross-validation scores: [0.8984985  0.89871612 0.89563781 0.89458668 0.90081838]
```

In [145…
```
# Print the mean accuracy and standard deviation
print(f"Mean Accuracy: {cv_scores.mean():.4f}")
print(f"Standard Deviation: {cv_scores.std():.4f}")
```

```
Mean Accuracy: 0.8977
Standard Deviation: 0.0023
```

As seen above, the cross-validation results show a mean accuracy of approximately 89.77% with a standard deviation of 0.0023. This suggests that the Random Forest model is performing consistently well across different subsets/folds of the data and the standard deviation is relatively low, indicating stability.

3 Conclusions

3.1 Evaluation

In summary, our classification approach has substantially improved the results, we started with a baseline of Logistic Regression with Count vectorizer which reached an accuracy of 79.9%, then using our best performing model, Random Forest with TF-IDF reached an outstanding accuracy of 90.5%.

Following that, RandomizedSearchCV was used to further improve the Random Forest model's performance. Due to resource and time constraints, using RandomizedSearchCV only increased the model's accuracy by 0.3%, the first run of RandomizedSearchCv with the Random Forest Classifier yielded 37% accuracy but after fine-tuning the hyperparameters and increasing the n_iter parameter in RandomizedSearchCV to increase the number of random combinations to try, an accuracy of 90.8% was achieved.
Finally, Cross-validation was used to assess the optimized model's performance to verify stability and consistency.

3.2 Evaluation of the project and its results

In summary, throughout this project proper preprocessing steps were conducted on the dataset, different vectorization methods were used, such as Count and TF-IDF vectorizers, to explore better-performing combinations, a proper baseline was chosen (Logistic Regression with Count Vectorizer), then, 3 models well-known for text-classification were implemented and compared, and the best-performing model and vectorization combination (Random Forest with TF-IDF Vectorization) was then used with RandomizedSearchCV to further increase performance, and confusion matrices were drawn consistently throughout the project to analyze the prediction summary in matrix form. Furthermore, Cross-validation was used as a final step to evaluate performance of the enhanced model ensuring consistency and stability.

Due to resources and time constraints, more hyperparameter tuning of the randomizedSearchCV wasn't conducted, such as increasing n_iter more, to expand the number of random combinations to try, which will result in longer computational time. However, as mentioned before great effors were made to get the best results given the circumstances and the randomizedSearchCV code was run twice with the same hyperparameters and it resulted in the same scores indicating stability and that the identified hyperparameters are likely to generalize well to unseen data. In addition, using more computationally expensive techniques that could yield better results, such as GridSearchCV wasn't possible.

Nevertheless, the major performance improvement (an increase of 10.9% in accuracy and weighted average) obtained within the given timeframe and considering resource constraints is extremely substantial.

References

[1] Twitter Sentiment Analysis notebooks on Kaggle: https://www.kaggle.com/datasets/jp797498e/twitter-entity-sentiment-analysis/code

[2] *Precision What is Precision?*, C3.ai. Link: https://c3.ai/glossary/machine-learning/precision/#:~:text=What%20is%20Precision%3F,the%20number%20of%20false%20positives.

[3] *What is recall in Machine Learning?*, Iguazio. Link: https://www.iguazio.com/glossary/recall/#:~:text=Recall%2C%20also%20known%20as%20the,total%20samples%

[4] Natasha Sharma, (2023, June 10). *Understanding and Applying F1 Score: A Deep Dive with Hands-On Coding*, Arize. Link: https://arize.com/blog-course/f1-score/#:~:text=F1%20score%20is%20a%20measure,better%20understanding%20of%20model%20performance.

[5] *Machine Learning - Performance Metrics*, Tutorialspoint. Link: https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_algorithms_performance_metr