Sentiment Analysis of Chat-GPT Twitter Reviews using BERT

By
Raghav Jindal

Introduction

Welcome to the world of sentiment analysis! In today's digital age, social media has become a powerful platform for expressing opinions and emotions. Twitter, being one of the most popular social media platforms, is an ideal source for capturing public sentiments on various topics.

Chat-GPT, a large language model, trained by OpenAI, is capable of generating human-like responses to text inputs. With the advent of Bidirectional Encoder Representations from Transformers (BERT), a state-of-the-art natural language processing model, sentiment analysis has been revolutionized. BERT has the ability to understand the context of the text and its relationships with other words in the sentence, making it an ideal candidate for sentiment analysis.

Objective

In this project, we will be using BERT to perform sentiment analysis on Twitter reviews generated by Chat-GPT. By analyzing the polarity of the reviews, we can determine whether the feedback is positive, negative, or neutral. This information can be used by businesses to make data-driven decisions and improve their products and services.

About the Dataset

- ChatGPT has been a major talk in the tech world. The tweets about chatgpt were gathered for a month and then the sentiment analysis was made using Natural Language Processing.
- The Dataset contains about 200000 Tweets and its labels whether the tweet is good , bad or neutral

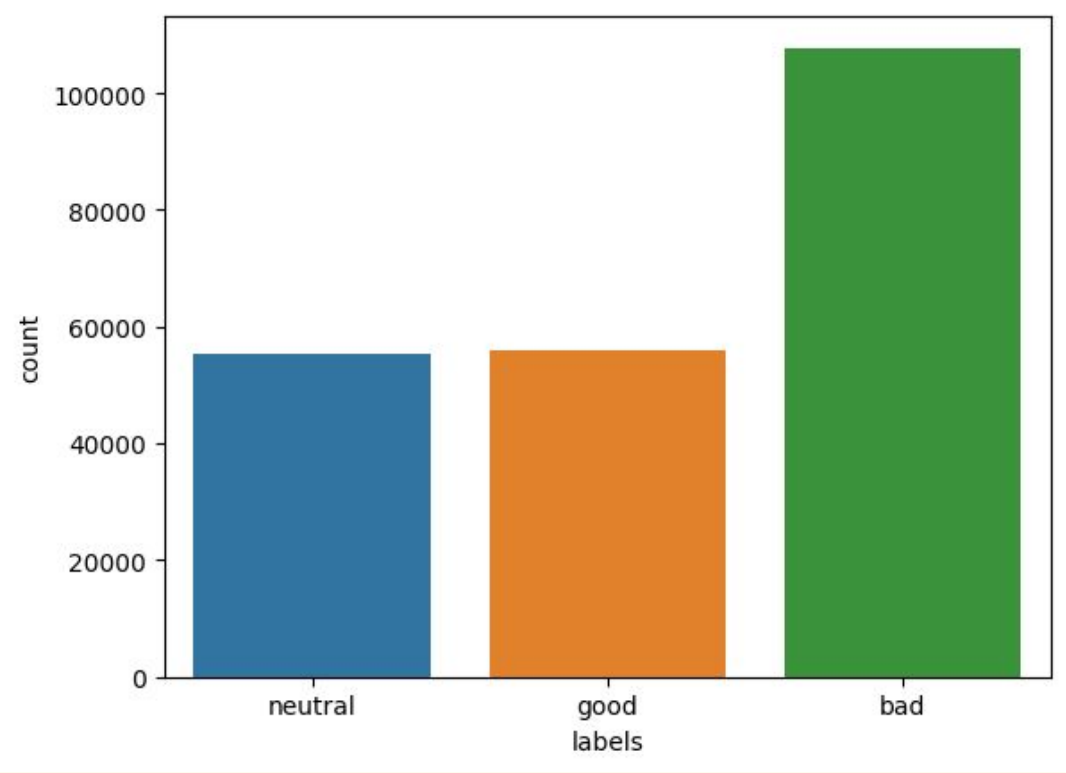
Procedure Followed

- **Step 1:** We install the transformer and import all the necessary packages.
- **Step 2:** We then, import the data and clean it
- **Step 3:** we import pre-trained DistilBERT model and tokenizer together.
- **Step 4:** Tokenize and process all sentences as batches.
- **Step 5:** Padding
- **Step 6: Masking:** Attention mask creates arrays of 0s(pad token) and 1s(real token).
- **Step 7:** Create an input tensor out of the padded token matrix, and send that to DistilBERT**.**
- **Step 8:** Train and test to evaluate performance of the model used.

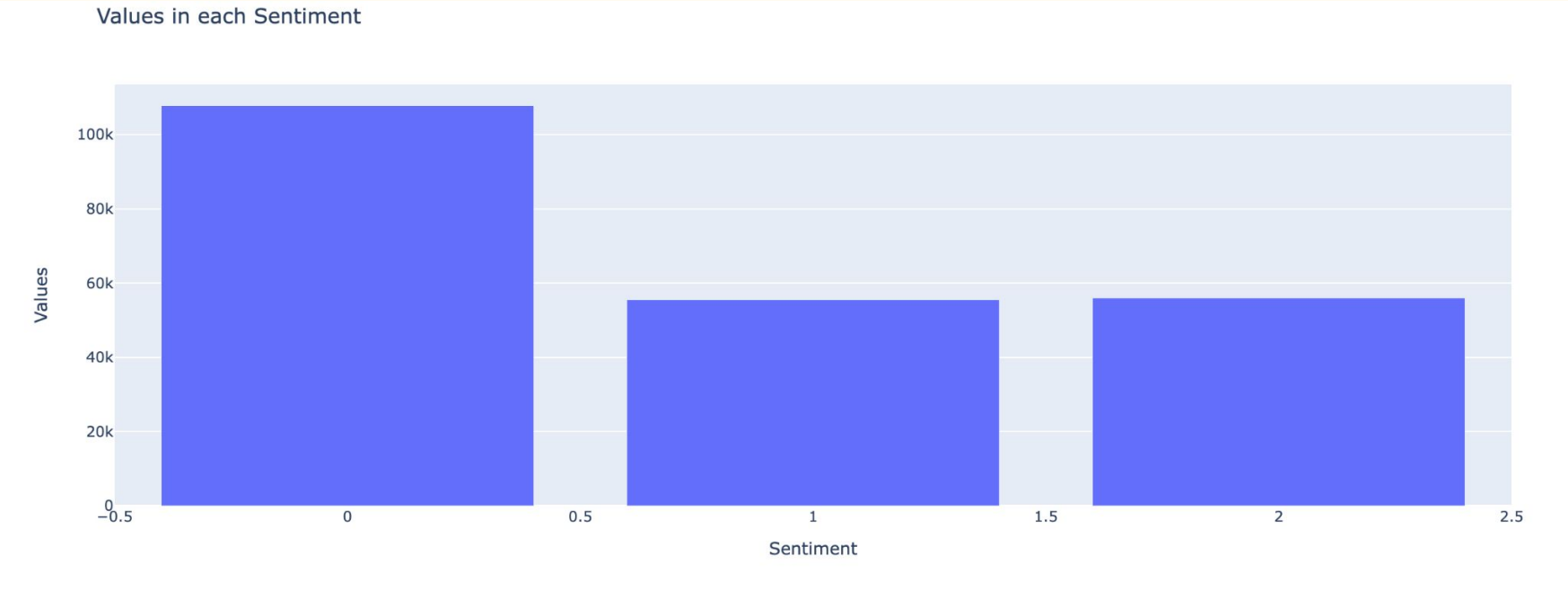
Models Used

- Gaussian Naive Bayes
- DecisionTreeClassifier
- K Nearest Neighbors Classifier
- Support Vector Machine
- Logistic Regression
- Random Forest Classifier

Data Visualization

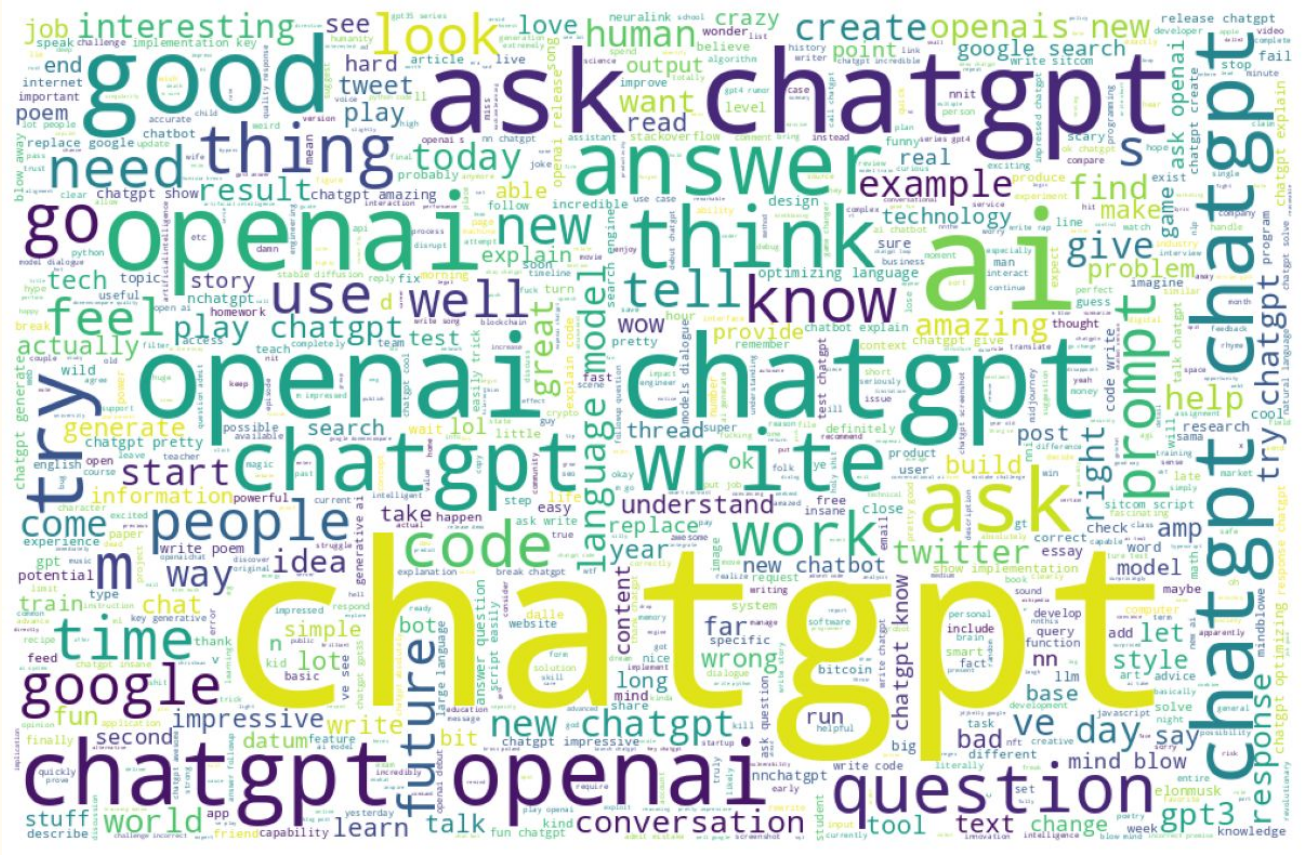


Data Visualization



Frequency of Word & Word Cloud (WHOLE)

	frequently_used_word	count
0	chatgpt	9320
1	openai	1613
2	ask	1217
3	ai	1203
4	write	1053
5	not	896
6	like	720
7	new	709
8	good	646
9	code	612
10	answer	608
11	question	587
12	try	545
13	google	510
14	m	458

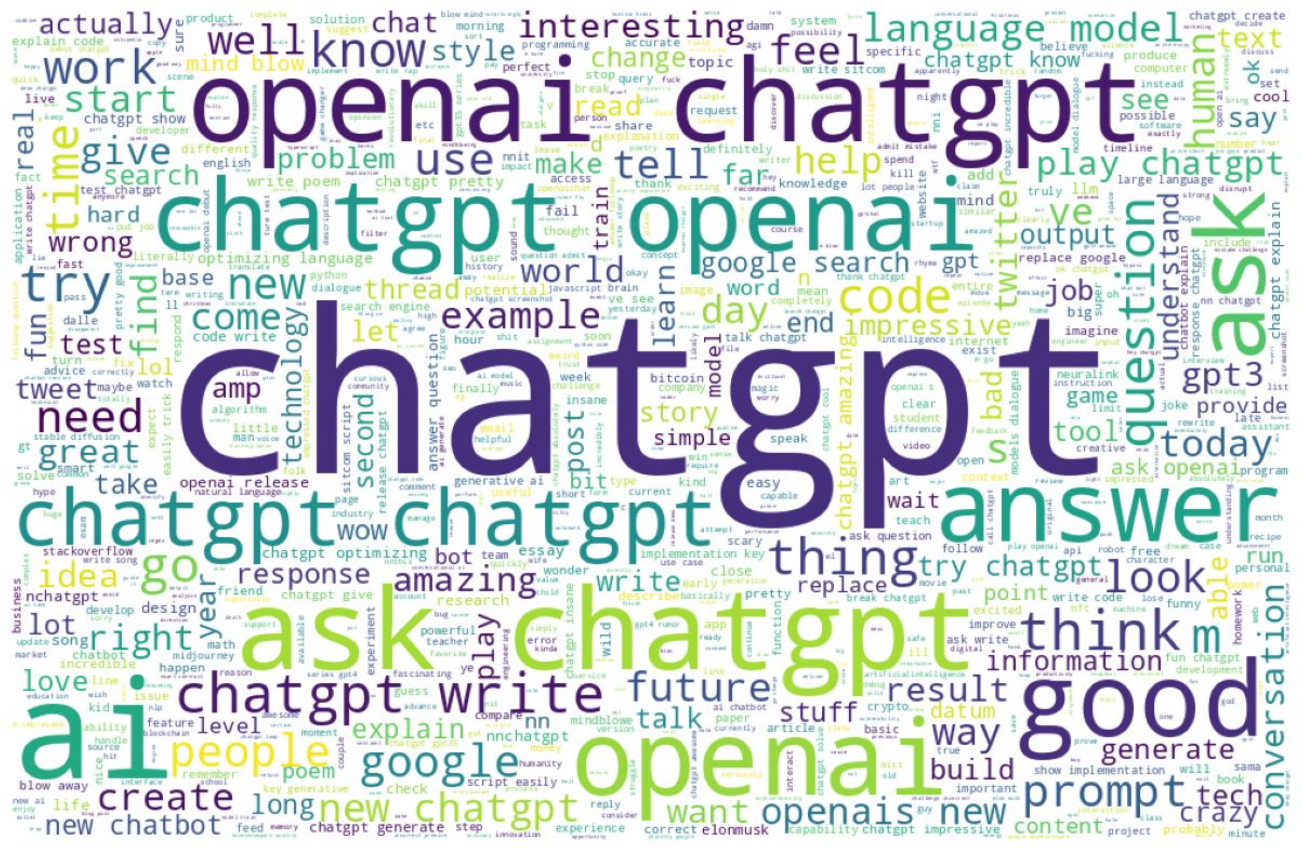


Frequency of Word & Word Cloud (POSITIVE)

	frequently_used_word	count
0	chatgpt	2647
1	openai	503
2	ai	475
3	good	423
4	like	419
5	ask	365
6	not	334
7	write	285
8	new	256
9	answer	249
10	question	221
11	try	212
12	m	204
13	amazing	204
14	impressive	203

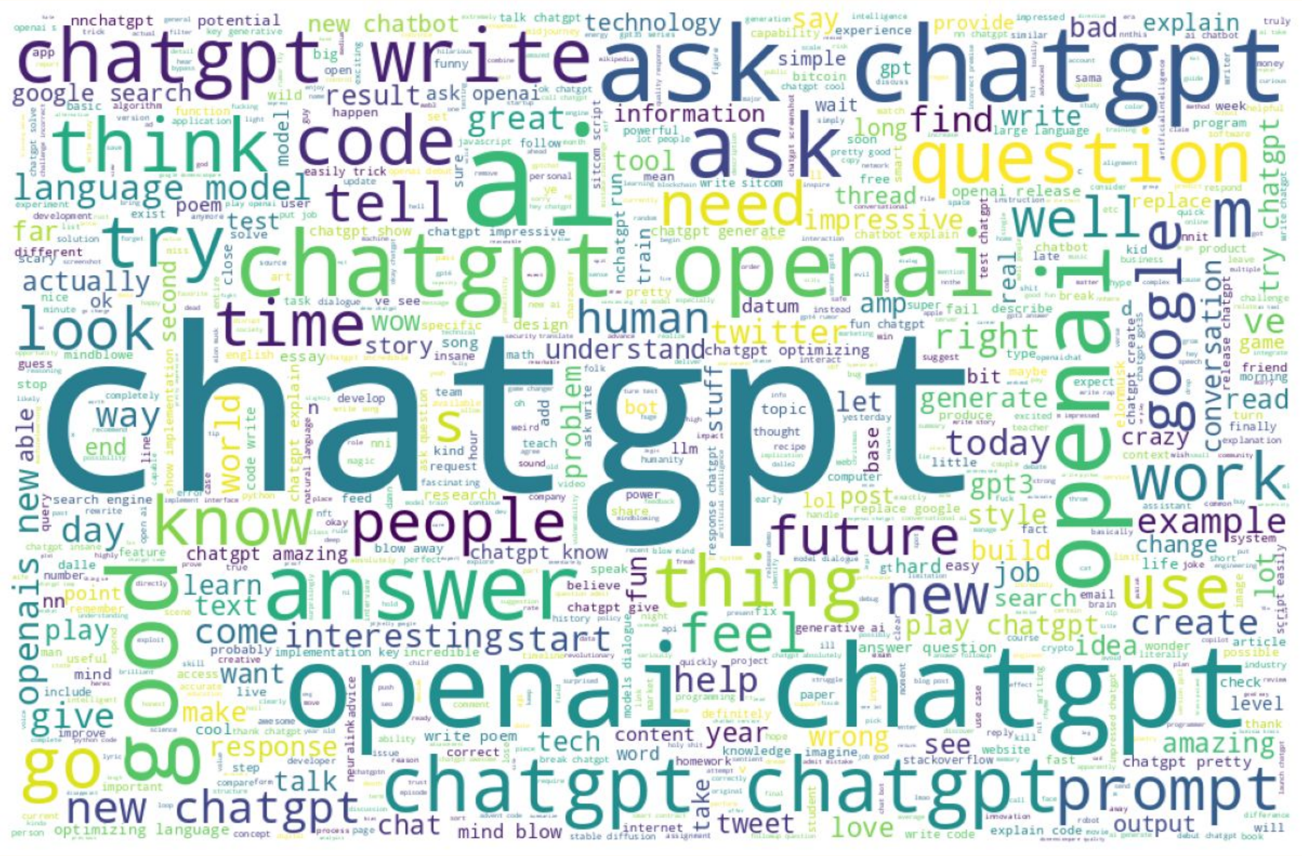
Frequency of Word & Word Cloud (NEUTRAL)

	frequently_used_word	count
0	chatgpt	2306
1	openai	417
2	ask	299
3	write	294
4	ai	262
5	like	224
6	not	217
7	new	213
8	code	194
9	good	169
10	openais	163
11	question	145
12	answer	143
13	try	137
14	language	135



Frequency of Word & Word Cloud (NEGATIVE)

	frequently_used_word	count
0	chatgpt	4367
1	openai	693
2	ask	553
3	write	474
4	ai	466
5	not	345
6	new	240
7	google	226
8	code	226
9	question	221
10	answer	216
11	try	196
12	go	192
13	think	191
14	know	182



Model Performance

	True Positive	False Positive	False Negative	True Negative	True Positive Rate	False Positive Rate	Accuracy
Gausian Naive Bayes	61	13	24	27	0.717647	0.325000	0.488
Decision Tree	67	20	17	21	0.797619	0.487805	0.536
K-NN	73	17	20	15	0.784946	0.531250	0.584
Linear SVM	75	16	21	13	0.781250	0.551724	0.600
Logisitic Regression	82	24	13	6	0.863158	0.800000	0.656
Random Forest Classifier	69	18	19	19	0.784091	0.486486	0.552

Model Performance

```
for model in models:
    scores = cross_val_score(model, test_features, test_labels, scoring='accuracy')
    print('Min Accuracy of {}: %.3f'.format(str(model)) % (scores.min()))
    print('Max Accuracy of {}: %.3f'.format(str(model)) % (scores.max()))
    print('Mean Accuracy of {}: %.3f'.format(str(model)) % (np.mean(scores)))
```

```
Min Accuracy of GaussianNB(): 0.600
Max Accuracy of GaussianNB(): 0.800
Mean Accuracy of GaussianNB(): 0.704
Min Accuracy of DecisionTreeClassifier(random_state=20): 0.560
Max Accuracy of DecisionTreeClassifier(random_state=20): 0.680
Mean Accuracy of DecisionTreeClassifier(random_state=20): 0.600
Min Accuracy of KNeighborsClassifier(): 0.600
Max Accuracy of KNeighborsClassifier(): 0.760
Mean Accuracy of KNeighborsClassifier(): 0.696
Min Accuracy of LogisticRegression(random_state=20): 0.680
Max Accuracy of LogisticRegression(random_state=20): 0.840
Mean Accuracy of LogisticRegression(random_state=20): 0.768
Min Accuracy of RandomForestClassifier(max_depth=4, random_state=20): 0.680
Max Accuracy of RandomForestClassifier(max_depth=4, random_state=20): 0.800
Mean Accuracy of RandomForestClassifier(max_depth=4, random_state=20): 0.728
Min Accuracy of LinearSVC(dual=False, random_state=20): 0.720
Max Accuracy of LinearSVC(dual=False, random_state=20): 0.800
Mean Accuracy of LinearSVC(dual=False, random_state=20): 0.752
```