Amazon Review Insight

A Sentiment Analysis & Summarization Project

Presented by: Teif Alharthi

TABLE OF CONTENTS

Project overview

Dataset Source:

Amazon Customer Reviews Dataset

Hypothesis:

NLP models can identify product sentiment and extract top insights for each product category.

Goal: Help customers make smarter purchase decisions.

Data Wrangling & Cleaning

- Removed duplicates, handled missing values, merged product categories.
- Used Transformer models to extract embeddings for clustering.
- Created new broad product categories (4-6 clusters).

Data Wrangling & Cleaning

Map star Ratings to sentiment classes

```
[ ] def map_star_to_sentiment(rating):
        if rating in [1, 2]:
            return 'NEGATIVE'
        elif rating == 3:
            return 'NEUTRAL'
        elif rating in [4, 5]:
            return 'POSITIVE'
```

```
df['true_sentiment'] = df['reviews.rating'].apply(map_star_to_sentiment)
```

```
[ ] print(df[['reviews.rating', 'true_sentiment']].head())
```

```
   reviews.rating true_sentiment
0               3        NEUTRAL
1               4       POSITIVE
2               5       POSITIVE
3               5       POSITIVE
4               5       POSITIVE
```

Data Wrangling & Cleaning

- ## Balance The Data

```
[ ] print(df['true_sentiment'].value_counts())
```

```
true_sentiment
POSITIVE    25545
NEGATIVE     1581
NEUTRAL      1206
Name: count, dtype: int64
```

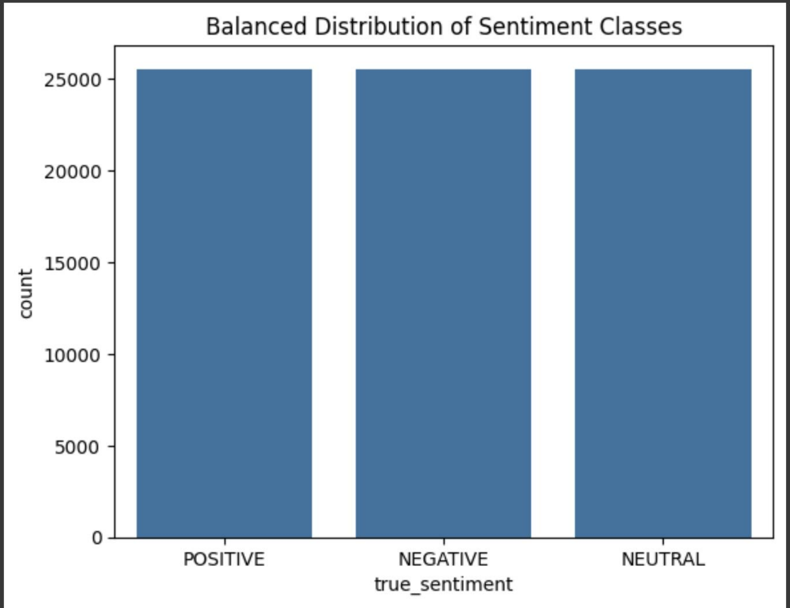
```
from sklearn.utils import resample
import pandas as pd


df_pos = df[df['true_sentiment'] == 'POSITIVE']
df_neg = df[df['true_sentiment'] == 'NEGATIVE']
df_neu = df[df['true_sentiment'] == 'NEUTRAL']


max_count = max(len(df_pos), len(df_neg), len(df_neu))


df_pos_upsampled = resample(df_pos, replace=True, n_samples=max_count, random_state=42)
df_neg_upsampled = resample(df_neg, replace=True, n_samples=max_count, random_state=42)
df_neu_upsampled = resample(df_neu, replace=True, n_samples=max_count, random_state=42)


df_balanced = pd.concat([df_pos_upsampled, df_neg_upsampled, df_neu_upsampled])


print(df_balanced['true_sentiment'].value_counts())
```

```
true_sentiment
POSITIVE    25545
NEGATIVE    25545
NEUTRAL     25545
Name: count, dtype: int64
```

Data Wrangling & Cleaning

- **Balance The Data**

```python
#true_sentiment
import seaborn as sns
import matplotlib.pyplot as plt
sns.countplot(x='true_sentiment', data=df)
plt.title('Balanced Distribution of Sentiment Classes')
plt.show()
```



Task

Classification with pre-trained models

01

First Model (bert-base-uncased)

[2517/2517 13:12, Epoch 3/3]

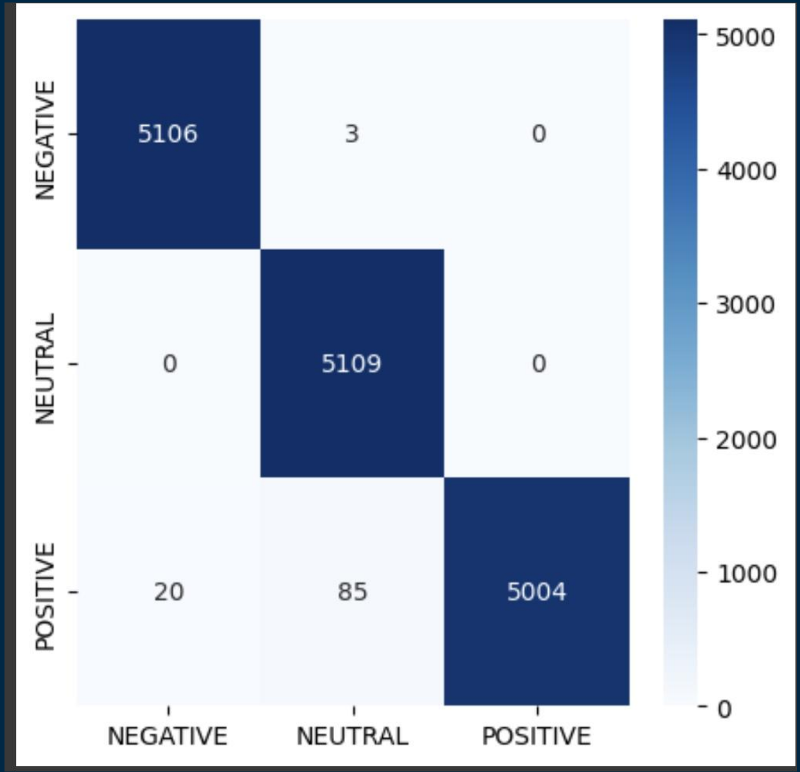
Epoch	Training Loss	Validation Loss	Accuracy	F1
1	0.267500	0.076729	0.978471	0.978511
2	0.044600	0.037582	0.989301	0.989288
3	0.019800	0.025546	0.993215	0.993208

```
preds_output = trainer.predict(emotion_encoded["test"])
preds_output.metrics
```

```
{'test_loss': 0.03169412538409233,
 'test_accuracy': 0.992953611274222,
 'test_f1': 0.9929438059711422,
 'test_runtime': 33.0649,
 'test_samples_per_second': 463.543,
 'test_steps_per_second': 7.258}
```

First Model (bert-base-uncased)

```
from sklearn.metrics import classification_report
print(classification_report(y_true, y_pred))

              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5109
           1       0.98      1.00      0.99      5109
           2       1.00      0.98      0.99      5109

    accuracy                           0.99     15327
   macro avg       0.99      0.99      0.99     15327
weighted avg       0.99      0.99      0.99     15327
```



First Model (bert-base-uncased)

```
id2label = {0: 'NEGATIVE', 1: 'NEUTRAL', 2: 'POSITIVE'}
pred_label = id2label[pred]
print(f"Predicted label: {pred_label}")
```

```
Text: I hated the product, it was terrible.
Predicted: NEGATIVE

Text: It was okay, nothing special.
Predicted: NEUTRAL

Text: I loved it! Best thing I've ever bought!
Predicted: POSITIVE

Predicted label: NEGATIVE
```

Second Model (Distilbert-base-uncased)

[10059/10059 30:17, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy	F1
1	0.092800	0.037796	0.989692	0.989690
2	0.025700	0.021342	0.993476	0.993471
3	0.009000	0.022656	0.994520	0.994519

```
[36] preds_output2 = trainer.predict(emotion_encoded["test"])
     preds_output2.metrics
```

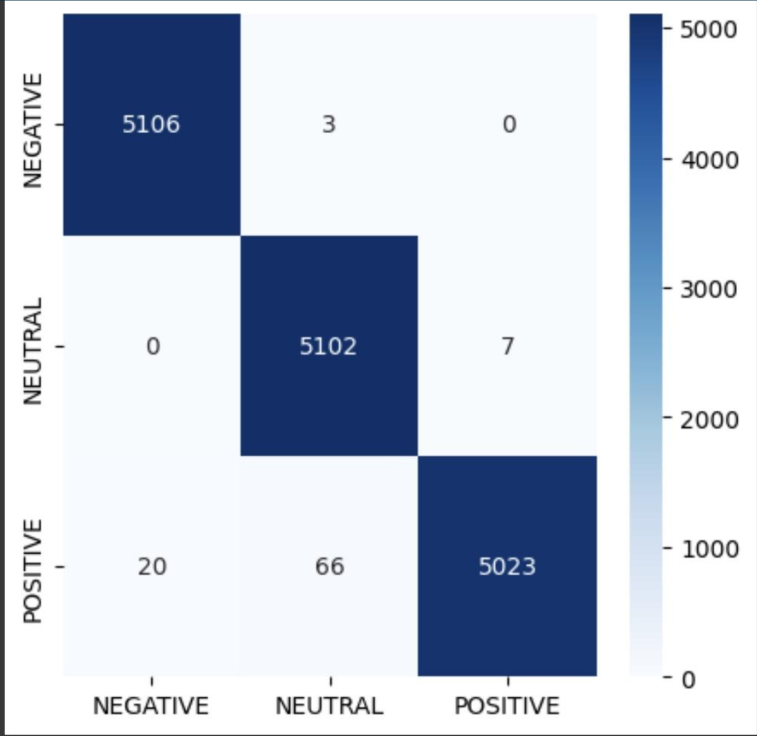
```
{'test_loss': 0.02696716971695423,
 'test_accuracy': 0.9937365433548639,
 'test_f1': 0.9937291035535536,
 'test_runtime': 55.5448,
 'test_samples_per_second': 275.939,
 'test_steps_per_second': 17.247}
```

Second Model (Distilbert-base-uncased)

```
[37] y_pred2 = np.argmax(preds_output2.predictions, axis=1)
     y_true2 = emotion_encoded["test"][:]["labels"]
```

```
from sklearn.metrics import classification_report
print(classification_report(y_true2, y_pred2))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5109
           1       0.99      1.00      0.99      5109
           2       1.00      0.98      0.99      5109

    accuracy                           0.99     15327
   macro avg       0.99      0.99      0.99     15327
weighted avg       0.99      0.99      0.99     15327
```



Third Model (DeBERTa-v3-base)

[2517/2517 17:52, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy	F1
1	0.842400	0.520803	0.799191	0.795776
2	0.452500	0.279792	0.907620	0.908472
3	0.284800	0.179658	0.945720	0.945705

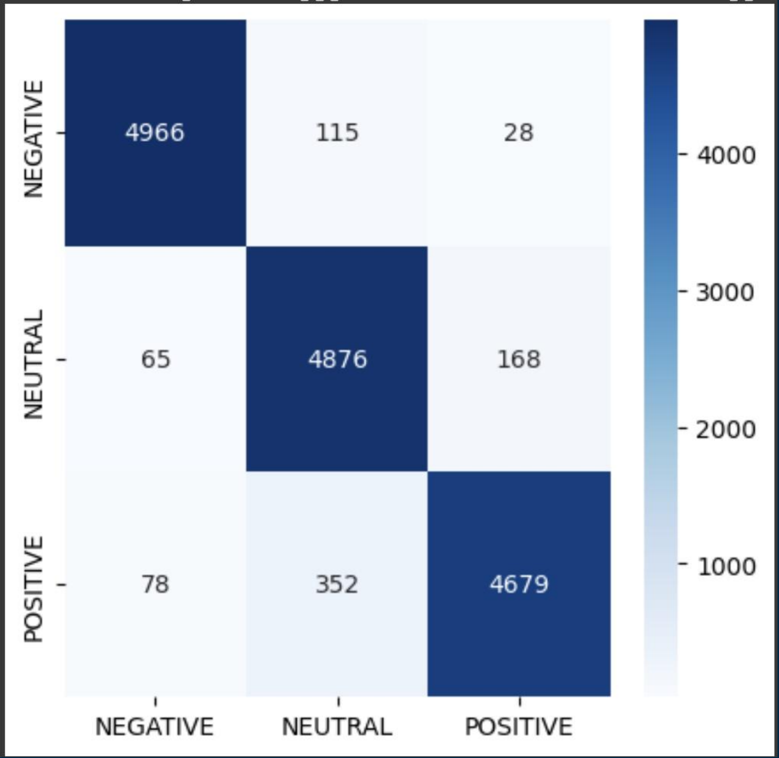
```
preds_output3 = trainer.predict(emotion_encoded["test"])
preds_output3.metrics
```

```
{'test_loss': 0.17642745375633 24,
 'test_accuracy': 0.9474130619168787,
 'test_f1': 0.9474456764791233,
 'test_runtime': 32.7124,
 'test_samples_per_second': 468.538,
 'test_steps_per_second': 7.337}
```

Third Model (DeBERTa-v3-base)

```
from sklearn.metrics import classification_report
print(classification_report(y_true3, y_pred3))
```

```
              precision    recall  f1-score   support

           0       0.97      0.97      0.97      5109
           1       0.91      0.95      0.93      5109
           2       0.96      0.92      0.94      5109

    accuracy                           0.95     15327
   macro avg       0.95      0.95      0.95     15327
weighted avg       0.95      0.95      0.95     15327
```



Performance Comparison and Chosen Model

My Choice:

I chose BERT-base-uncased because it gave the best overall performance and the most accurate results.

First Model

Achieved the highest accuracy (99%) with almost perfect precision and recall. Very strong performance across all sentiment classes

Second Model

Also achieved 99% accuracy, but with slightly more errors. It's a lighter and faster version of BERT with good performance

Third Model

Had the lowest accuracy (95%) and more misclassifications, especially in the positive class.

Task

Product Category Clustering

02

```
# 2. Generate embeddings using a transformer model
!pip install -U sentence-transformers

from sentence_transformers import SentenceTransformer

model = SentenceTransformer('all-MiniLM-L6-v2')

embeddings = model.encode(categories)
```

Show hidden output

```
# 3. Apply clustering (KMeans)
from sklearn.cluster import KMeans

k = 4
kmeans = KMeans(n_clusters=k, random_state=0)
cluster_labels = kmeans.fit_predict(embeddings)
```


```
# 3. Apply clustering (KMeans)
from sklearn.cluster import KMeans

k = 4
kmeans = KMeans(n_clusters=k, random_state=0)
cluster_labels = kmeans.fit_predict(embeddings)
```


```
from collections import defaultdict

cluster_map = defaultdict(list)
for cat, label in zip(categories, cluster_labels):
    cluster_map[label].append(cat)


for label, items in clu (variable) label: Any
    print(f"\nCluster {label}:")
    for item in items:
        print("  -", item)
```

Show hidden output

```
category_to_cluster = dict(zip(categories, cluster_labels))
df['category_cluster'] = df['categories'].map(category_to_cluster)
```

```
cluster_names = {
    0: 'Fire & Amazon Tablets',
    1: 'eBook Readers & Accessories',
    2: 'Home, Health & Office Essentials',
    3: 'Smart Home & Entertainment Devices'
}


df['meta_category'] = df['category_cluster'].map(cluster_names)
```

```
df['meta_category'].value_counts()
```

meta_category	count
Fire & Amazon Tablets	14297
Smart Home & Entertainment Devices	12732
eBook Readers & Accessories	1210
Home, Health & Office Essentials	93

```
threshold = 1000
category_counts = df['meta_category'].value_counts()

def merge_small_clusters(df, category_counts, threshold):

    category_merge = {}

    for category, count in category_counts.items():
        if count < threshold:

            max_category = category_counts.idxmax()
            category_merge[category] = max_category
            category_counts[max_category] += count
        else:
            category_merge[category] = category

    df['merged_category'] = df['meta_category'].map(category_merge)
    return df


df_merged = merge_small_clusters(df, category_counts, threshold)

print(df_merged[['meta_category', 'merged_category']].head())
```

```
                        meta_category                     merged_category
0  Smart Home & Entertainment Devices  Smart Home & Entertainment Devices
1  Smart Home & Entertainment Devices  Smart Home & Entertainment Devices
2  Smart Home & Entertainment Devices  Smart Home & Entertainment Devices
3  Smart Home & Entertainment Devices  Smart Home & Entertainment Devices
4  Smart Home & Entertainment Devices  Smart Home & Entertainment Devices
```

```python
new_category_counts = df_merged['merged_category'].value_counts()

print(new_category_counts)
```

```
merged_category
Fire & Amazon Tablets                    14390
Smart Home & Entertainment Devices       12732
eBook Readers & Accessories               1210
Name: count, dtype: int64
```

Why 3 Meta-Categories Instead of 4–6?

Although the original objective suggested 4–6 categories, I found that:

- Some clusters had very few samples and lacked meaningful patterns.
- Keeping them would reduce the clarity and usability of the results.
- I applied a merging step to group small clusters into the most similar larger ones.

Final Result:

- 3 strong and interpretable meta-categories
- Better data balance and cleaner grouping

Reasoning:
Quality over quantity — fewer, well-defined categories are more useful than weakly separated ones.

Task

03

Summarize reviews using generative AI

```python
category_name = 'Fire & Amazon Tablets'
category_df = df[df['merged_category'] == category_name]
```

```python
product_scores = category_df.groupby('name')['reviews.rating'].agg(['mean', 'count']).reset_index()
top_3 = product_scores.sort_values(['mean', 'count'], ascending=[False, False]).head(3)
worst_product = product_scores.sort_values('mean', ascending=True).iloc[0]
```

```python
from transformers import pipeline
summarizer = pipeline("summarization", model="facebook/bart-large-cnn", device=0)
```

Show hidden output

```python
def summarize_reviews(product_name):
    reviews = category_df[category_df['name'] == product_name]['reviews.text'].dropna().tolist()
    reviews_text = " ".join(reviews[:20])
    if len(reviews_text) < 50:
        return "Not enough review text."
    summary = summarizer(reviews_text, max_length=300, min_length=100, do_sample=False)[0]['summary_text']
    return summary
```

```python
summary1 = summarize_reviews(top_3.iloc[0]['name'])
summary2 = summarize_reviews(top_3.iloc[1]['name'])
summary3 = summarize_reviews(top_3.iloc[2]['name'])
```

🔥 Category: Smart Home Devices

🎉 Welcome to the Ultimate Smart Home Devices Review!

Here's a quick look at the top 3 standout products in this category, a quick comparison to help you decide, and our top recommendation for what *not* to buy!

📌 Top 3 Products:

Product 1: Echo Dot (5th Gen)

Summary: Great value, strong Alexa integration, and compact design. **Top Complaint**: Voice recognition in noise **Key Strength**: Smart assistant use

---

Product 2: Ring Video Doorbell 4

Summary: High-quality video with reliable motion alerts and easy setup. **Top Complaint**: App glitches **Key Strength**: Home security

---

Product 3: TP-Link Kasa Smart Plug

Summary: Affordable and works seamlessly with major smart ecosystems. **Top Complaint**: Outdated app interface **Key Strength**: Budget automation

---

📊 Quick Comparison:

Product	Best For	Key Complaint
Echo Dot (5th Gen)	Smart assistant use	Voice recognition in noise
Ring Video Doorbell 4	Home security	App glitches
TP-Link Kasa Smart Plug	Budget automation	Outdated app interface

---

👎 Worst Product: Unknown Brand Smart Plug

Average Rating: 2.3 / 5 **Why It Falls Short**: Connection drops frequently, and app is full of bugs. **Top Complaints**:

- Unreliable connectivity
- Difficult setup process
- Poor customer support

🌟 Conclusion

In conclusion, **Smart Home Devices** offers a wide variety of smart solutions, but not all are equal. Stick to reputable brands for reliability and better user experience, and always check reviews before buying—especially with unknown or generic brands.

Model Deployment on AWS

Deployment Summary:

- Model: BERT fine-tuned for review classification
- Platform: Deployed on AWS EC2 instance
- Interface: Built using Streamlit
- Access: Public web interface available

Why EC2?

- Offers flexibility and control over deployment environment
- Suitable for hosting ML models with real-time inference

How it works:

1. User enters a product review
2. The model processes the text using BERT
3. It returns the sentiment classification (positive, negative, neutral)

http://51.20.87.52:8501/

THANK YOU