Project NLP | Business Case: Automated Customer

Group3:

-Edward Cruz

-Alex Castillo

-Joshua Lopez

-Gerardo Gonzalez

Executive Summary

This business case outlines the development of an NLP model to automate the processing of customer feedback for a retail company. The goal is to classify customer reviews into positive, negative, or neutral categories to help the company improve its products and services. The second part is to GenerativeAI to summarize reviews broken down into review score (0-5), and broken down into product categories - if the categories are too many to handle, select a top-K categories. Create a clickable and dynamic visualization dashboard using a tool like Tableau, Plotly, or any of your choice.

Project goals

- The ML/AI system should be able to run classification of customers' reviews (the textual content of the reviews) into positive, neutral, or negative.
- For a product category, create a summary of all reviews broken down by each star or rating (we should have 5 of these).
    - If your system can't handle all products categories, pick a number that you can work with (eg top 10, top 50, Etc)

1)Traditional NLP & ML approaches

- Dateset used: 1429_1.csv

  From: https://www.kaggle.com/datasets/datafiniti/consumer-reviews-of-amazon-products

(a) Data Pre-processing:

- We analyzed the dataset to determine which columns were really useful for the case.
- Check features that had null values and clean them.
- After removing columns tat had most null values. By checking the distributions of the other features we determined that for the sentiment classifier, the most important feature are “reviews.text” (model input features) and “reviews.rating” (model target) .

Noted: From the analysis we found that the dataset is very unbalance.

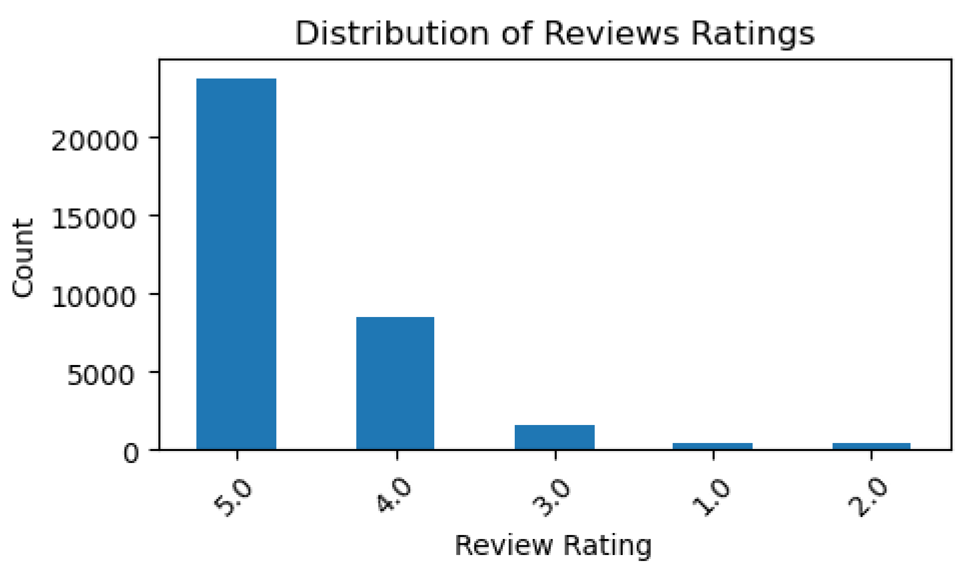

Figure 1: Original Data Ratings Distribution

	reviews.rating	No of Users
0	5.0	23775
1	4.0	8541
2	3.0	1499
3	1.0	410
4	2.0	402

(b) Texts Tokenize and Remove stopwords

i) First split the dataset into features and target

ii) To the input feature column (review.tex):
   - Clean stop words (NLTK)
   - Convert to lowercase
   - Remove punctuations

iii) To the target column (reviews.rating):
   - creates a function to map rating to sentiments.
   - Positive (ratings 4-5)
   - Negative (ratings 1-2)
   - Neutral (ratings 3)

(c) Vectorize text data using TF-IDF and Balancing

-Vectorize the clean review.text.

- I apply SMOTE on the vectorize data to balance the dataset.

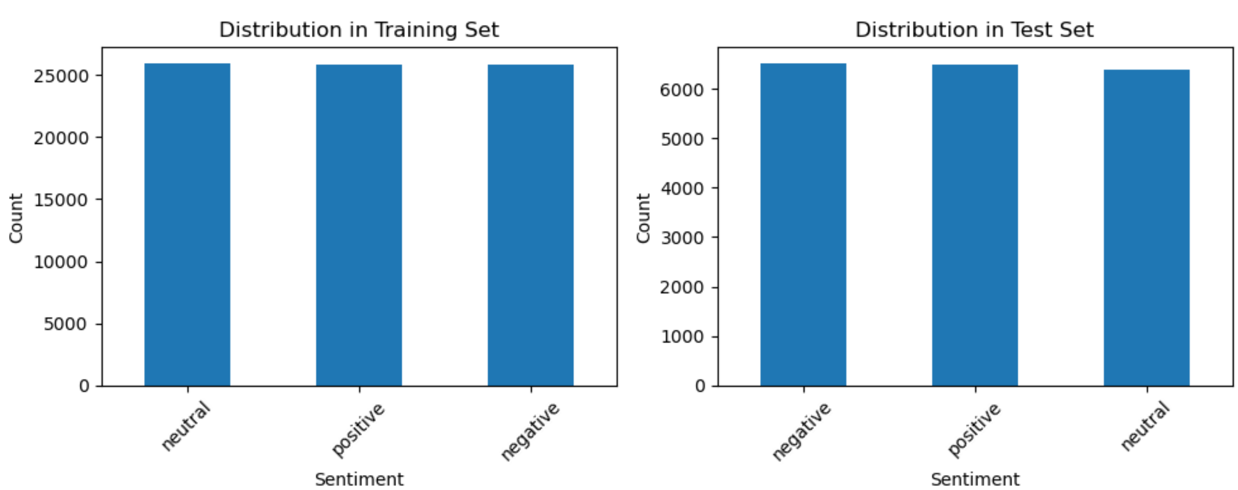

Figure 2: Distribution after grouping and Balancing

(d) Split data into Training and Test

-Train 80%

-Test 20%

(e)Train using Different Traditional NLP and ML models

1) Naive Bayes:

```
Accuracy: 0.84
Classification Report:
              precision    recall  f1-score   support

    negative       0.86      0.89      0.87      6510
     neutral       0.80      0.77      0.79      6388
    positive       0.86      0.86      0.86      6488

    accuracy                           0.84     19386
   macro avg       0.84      0.84      0.84     19386
weighted avg       0.84      0.84      0.84     19386
```

Figure 3: Naive Bayes Classification Report

```
Confusion Matrix:
[[5775  495  240]
 [ 793 4916  679]
 [ 179  701 5608]]
```

Figure 4: Confusion Matrix

2)Gradient Boosting

Report:

```
Accuracy: 0.66

Gradient Boosting Classification Report:
              precision    recall  f1-score   support

    negative       0.77      0.68      0.72      6510
     neutral       0.52      0.71      0.60      6388
    positive       0.78      0.61      0.68      6488

    accuracy                           0.66     19386
   macro avg       0.69      0.66      0.67     19386
weighted avg       0.69      0.66      0.67     19386
```

Figure 5: Evaluations Classification Report Scores

Confusion Matrix:

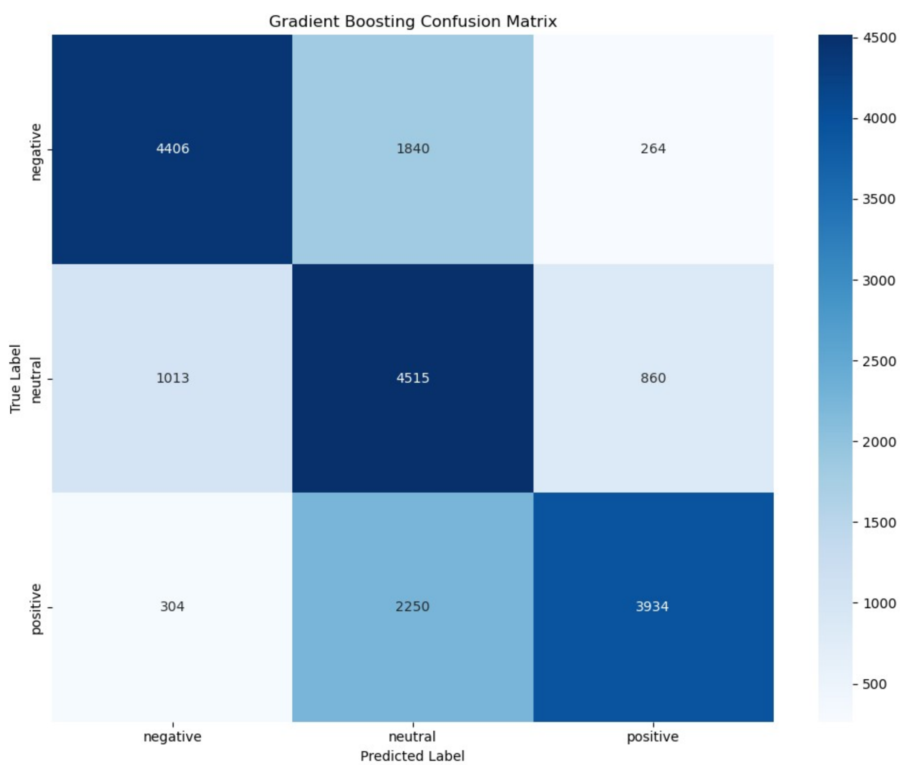

Figure 6: Confusion Matrix

3)XGBoost

Report:

XGBoost Classification Report:

	precision	recall	f1-score	support
negative	0.87	0.83	0.85	6510
neutral	0.80	0.74	0.77	6388
positive	0.78	0.88	0.83	6488
accuracy			0.82	19386
macro avg	0.82	0.82	0.81	19386
weighted avg	0.82	0.82	0.82	19386

Figure 7: Classification Report Scores

Confusion Matrix:

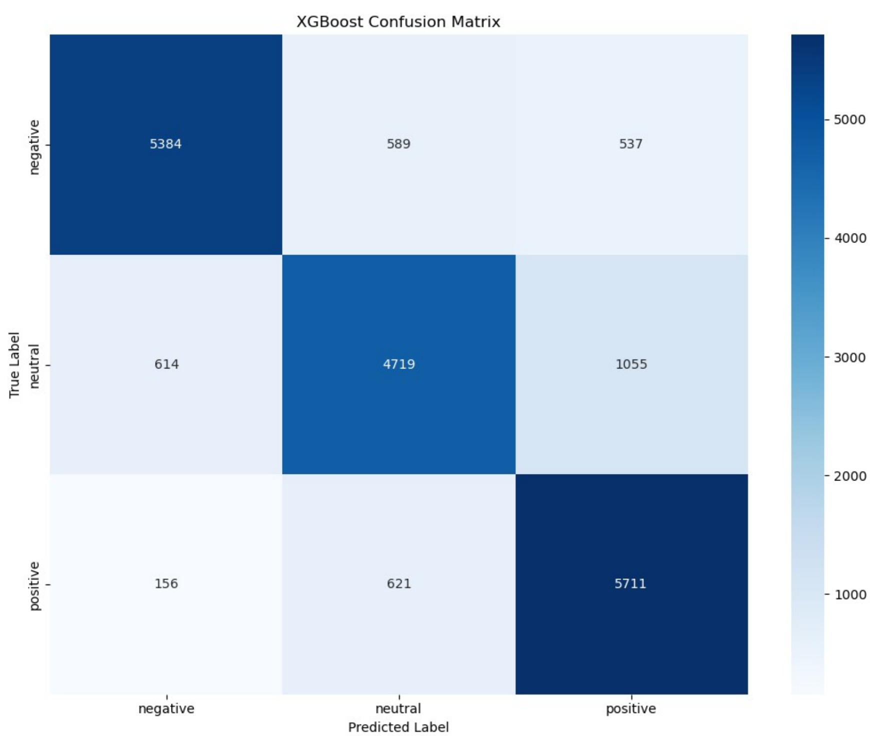

Figure 8: Confision Matrix

4)Random Forest L2 Regularize

Parameter:

```python
rf_regularized = RandomForestClassifier(
    n_estimators=100,
    max_depth=None,                # Limit tree depth
    min_samples_split=5,          # Require more samples to split
    min_samples_leaf=4,           # Require more samples in leaves
    max_features='sqrt',          # Reduce features considered per split
    max_samples=0.8,             # Use bootstrapping with 80% of samples
    random_state=42
)
```

Report:

```
Accuracy: 0.91

Random Forest Classification Report:
              precision    recall  f1-score   support

    negative       0.94      0.95      0.95      6510
     neutral       0.92      0.87      0.89      6388
    positive       0.88      0.93      0.91      6488

    accuracy                           0.91     19386
   macro avg       0.92      0.91      0.91     19386
weighted avg       0.92      0.91      0.91     19386
```

Figure 9: Evaluation Classification Report Scores

Confusion Matrix:

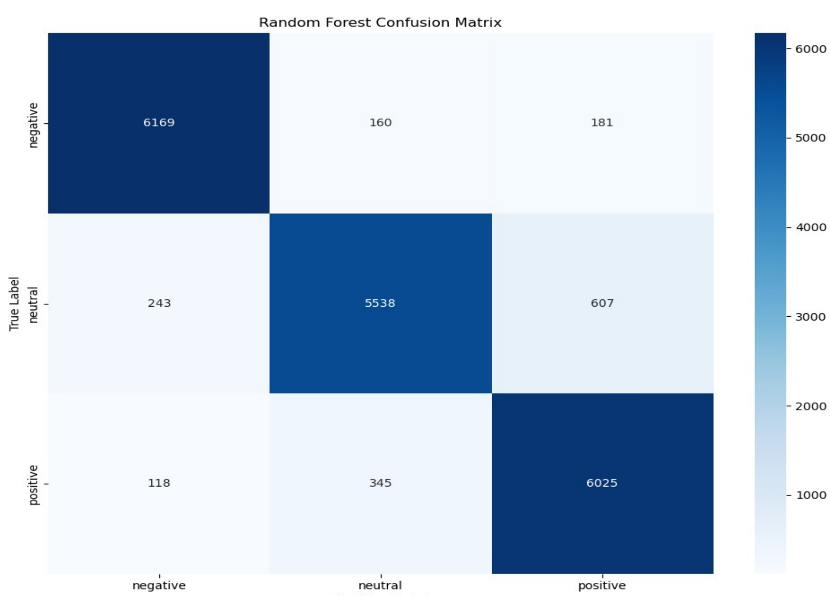

Figure 10: Confusion Matrix

Cross-Validation:

```
Cross-validation scores: [0.89941324 0.90346918 0.90063193 0.9048878  0.89760124]
Average CV score: 0.9012006781887552
CV score standard deviation: 0.0026538770363785692
```

Figure 11: Cross - Validation

5) Random Forest (BEST)

Parameter:

```
# Create and train the Random Forest model
rf_classifier = RandomForestClassifier(
    #class_weight='balanced',
    criterion='gini',
    n_estimators=100,  # number of trees
    max_depth=None,    # maximum depth of trees
    min_samples_split=2,
    min_samples_leaf=1,
    random_state=42,
    warm_start = False,
    verbose = 1    # for reproducibility
)
```

Report:

```
Accuracy: 0.99

Random Forest Classification Report:
              precision    recall  f1-score   support

    negative       0.99      0.99      0.99      6510
     neutral       0.98      0.99      0.98      6388
    positive       0.99      0.98      0.99      6488

    accuracy                           0.99     19386
   macro avg       0.99      0.99      0.99     19386
weighted avg       0.99      0.99      0.99     19386
```

Figure 12: Random Forest Classification Report Evaluation

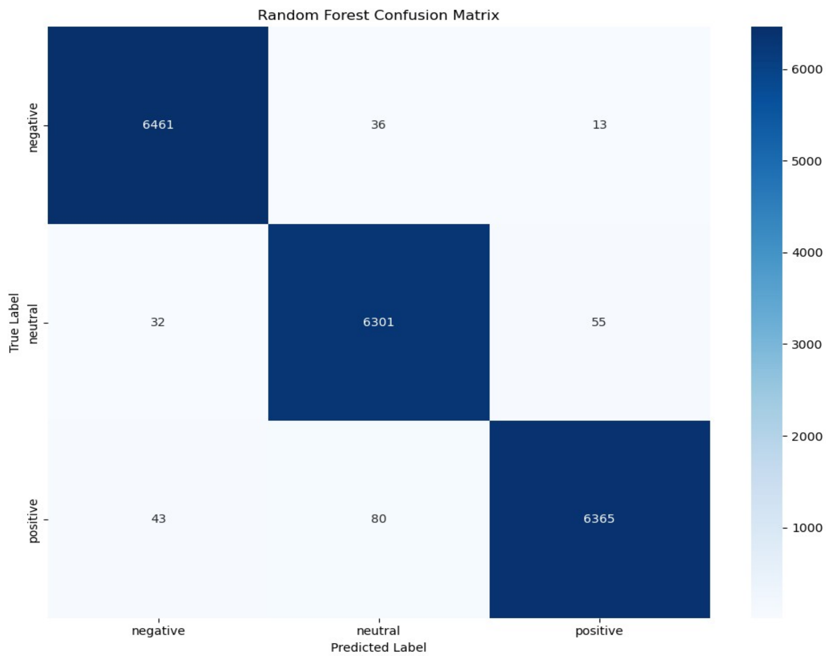

Figure 13: Confusion Matrix

```
Cross-validation scores: [0.98084983 0.98084859 0.98091308 0.98284756 0.9800748 ]
Average CV score: 0.981106772635683
CV score standard deviation: 0.0009236352971816066
```

Figure 14: Cross Validation Analysis

Analysis:

#These cross-validation results:

1. Consistency across folds:

- The scores across all 5 folds are very consistent (ranging from 0.980 to 0.982)
- The standard deviation is very small (0.00092), which is excellent
- This consistency suggests that your model is stable and performs similarly across different subsets of the data

2. Average CV Score:

- Average score of 0.981 (98.1%) is slightly lower than your test set accuracy of 0.99
- This small difference (about 0.9%) between CV and test performance suggests that your model isn't severely overfitting
- It's performing consistently well across different data splits

3. Analysis:

- The high scores with low variance suggest your model is genuinely learning the patterns in your data
- The small gap between CV scores and test accuracy indicates minimal overfitting

- The extremely low standard deviation (0.00092) shows remarkable stability in model performance

Conclusion:
Based on these cross-validation results, I would say that the model is NOT overfitting.
The reasons are:
1. Very stable performance across different data splits
2. Small difference between CV and test performance
3. Extremely low standard deviation in CV scores

This Random Forest model appears to be genuinely good at this classification task, likely because:
- The relationship between review text and sentiment might be relatively straightforward
- You probably have a good feature representation
- The classes are well-balanced after your preprocessing

2) **Sequence-to-Sequence modeling with LSTM**

- Goal: Build a Biderectional LSTM model to predict the review class i.e., negative, positive, or neutral.

a) I did the same data preprocessing as for the traditional NPL models:
    - Deleted unnecessary columns
    - Remove nulls
    - Remove stop-word, punctuation.
    - Convert to lowercase
    - Word tokenization
    - Vectorize the data
    - Balance the data with SMOTE
    - Split data into training and validation datasets

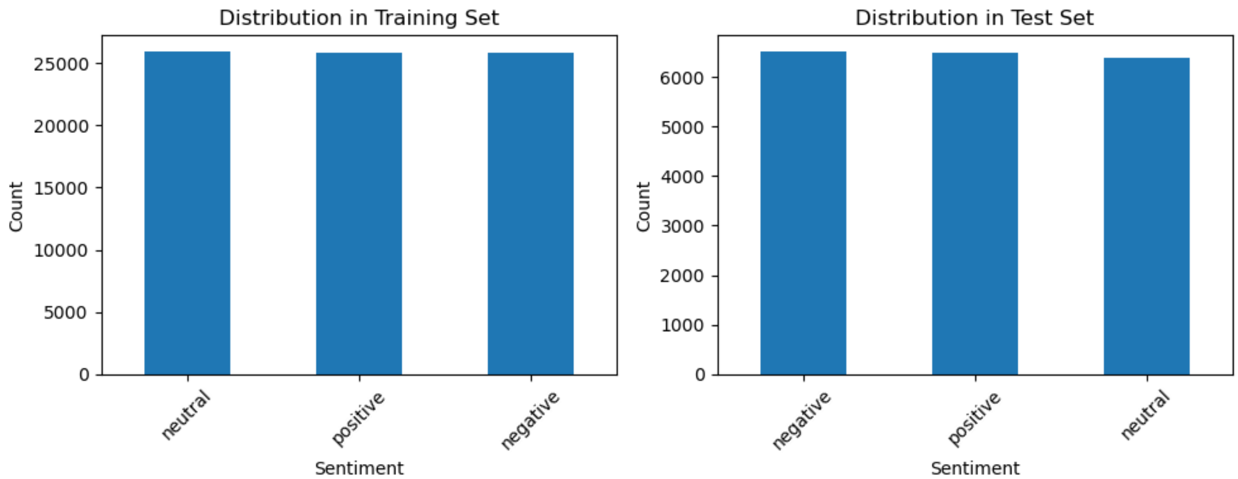

Figure 15: Distribution after Balancing Data

b) Train LSTM MODEL

- Reshape input data for LSTM (samples, timesteps, features)
- Architecture:

```
# Create LSTM Model
model = Sequential([
    LSTM(64, input_shape=(timesteps, input_dim), return_sequences=False),
    Dropout(0.5),
    Dense(32, activation='relu'),
    Dropout(0.5),
    Dense(3, activation='softmax')  # 3 classes: negative, neutral, positive
])
```

```python
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)


# Add early stopping
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)
```

```python
# Train the model
history = model.fit(
    X_train_reshaped,
    y_train,
    epochs=20,
    batch_size=64,
    validation_split=0.2,
    callbacks=[early_stopping]
)
```

c) Evaluation

Report:

```
Overall Accuracy: 94.82%

Classification Report:
              precision    recall  f1-score   support

    Negative       0.95      1.00      0.97      6462
     Neutral       0.92      0.96      0.94      6462
    Positive       0.98      0.88      0.93      6462

    accuracy                           0.95     19386
   macro avg       0.95      0.95      0.95     19386
weighted avg       0.95      0.95      0.95     19386
```

Figure 16: Evaluation Report

Confusion Matrix:

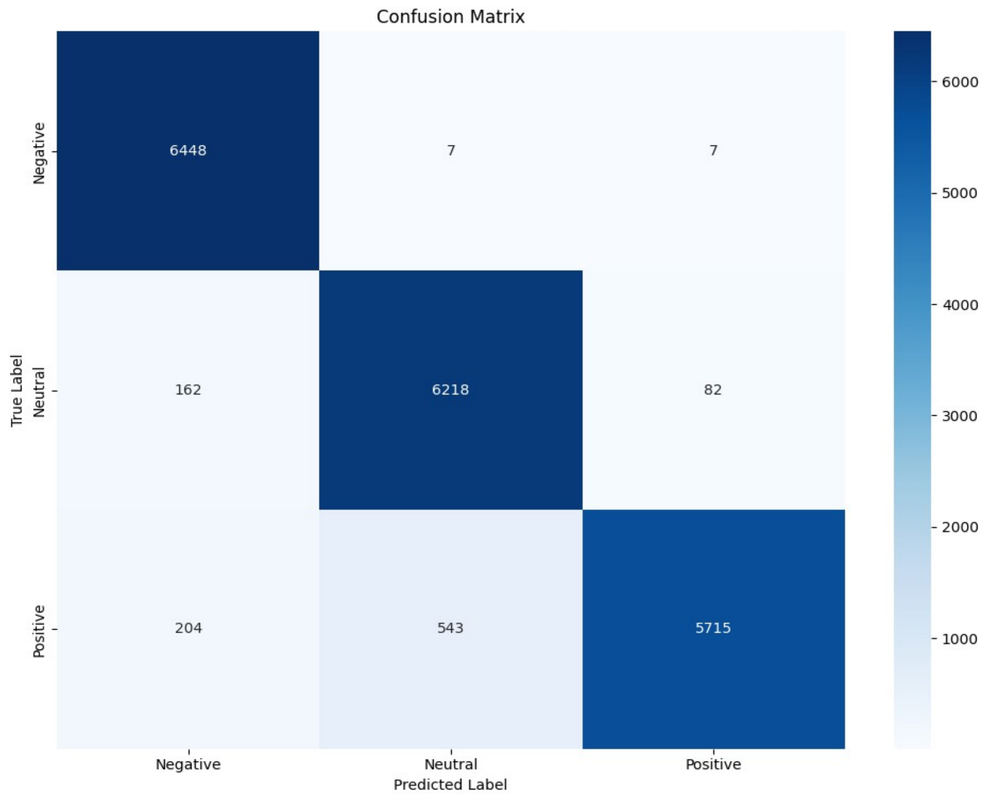

Figure 17: Confusion Matrix

Accuracy and Loss plots:

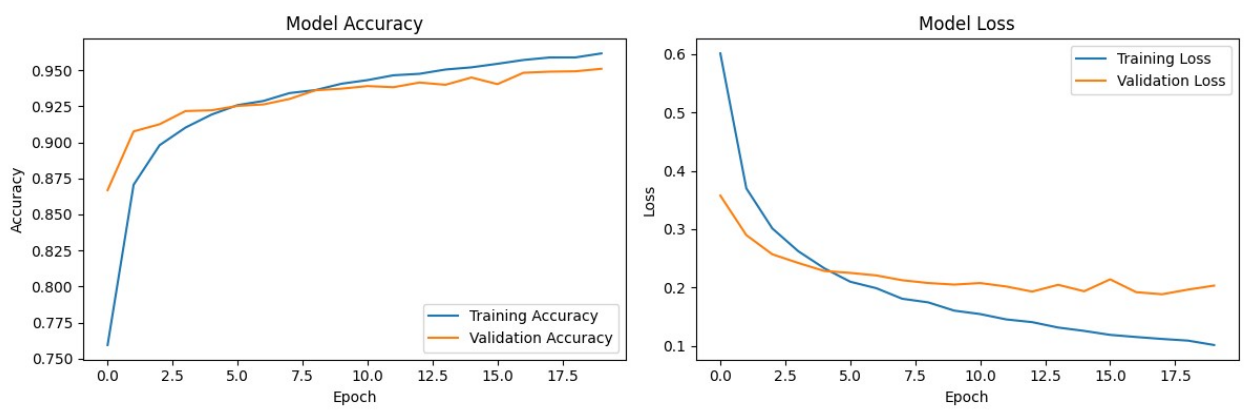

Figure 18: Loss and Accuracy Plots

3) Transformer approach (HuggingFace API) for Summarys

The goal was to have **a summary of all reviews broken down by each star or rating, for all product categories**. It would look like this, where every product had, for each rating, a single summarization of all reviews combined of that rating.

Product Category	Stars	Summaries of reviews
Electronics	☆	Summary of all 1 star reviews
	☆☆	Summary of all 2 star reviews
	☆☆☆	Summary of all 3 star reviews
	☆☆☆☆	Summary of all 4 star reviews
	☆☆☆☆☆	Summary of all 5 star reviews
Electronics,Media	☆	Summary of all 1 star reviews
	☆☆	Summary of all 2 star reviews
	☆☆☆	Summary of all 3 star reviews

The dataset used:
Datafiniti_Amazon_Consumer_Reviews_of_Amazon_Products.csv
from https://www.kaggle.com/datasets/datafiniti/consumer-reviews-of-amazon-products

```
kaggle_df = pd.read_csv('Datafiniti_Amazon_Consumer_Reviews_of_Amazon_Products.csv') # Load Dataset
```

The model used: **t5-small** from https://huggingface.co/google-t5/t5-small

```
# Initialize the model and tokenizer
model_name = "t5-small" # model
model = T5ForConditionalGeneration.from_pretrained(model_name, device_map={"": 0})

tokenizer = T5Tokenizer.from_pretrained(model_name)
```

1) Foundation Model

a)Data Preprocessing

1) Data Cleaning and Tokenization

-Cleaning NULLS

Since the columns that were going to be used were **“categories”, “review.rating”** and **“review.text”**, and they didn’t have **NULL** values, eliminating the **NULL** fields was not needed.

Clean Nulls

```
print(kaggle_df.isnull().sum()) # No NULLs found in categories, reviews.text or reviews.rating
```

✓ 0.0s

```
id                        0
dateAdded                 0
dateUpdated               0
name                      0
asins                     0
brand                     0
categories                0
primaryCategories         0
imageURLs                 0
keys                      0
manufacturer              0
manufacturerNumber        0
reviews.date              0
reviews.dateAdded      3948
reviews.dateSeen          0
reviews.doRecommend       0
reviews.id             4971
reviews.numHelpful        0
reviews.rating            0
reviews.sourceURLs        0
reviews.text              0
reviews.title            13
reviews.username          1
sourceURLs                0
dtype: int64
```

Figure 19: Dataset information

b)Tokenization

For the tokenization the T5 library provided an instruction to encode the data to be summarized.

```
tokenizer = T5Tokenizer.from_pretrained(model_name)

# Join reviews into a single string
text = "\n\n".join(dataframe)

# Tokenize and summarize the input text. inputs is a pytorch tensor, torch.Tensor
inputs = tokenizer.encode("summarize: " + text, return_tensors = "pt", truncation = True).to("cuda:0")
```

Figure 20: Tokenizer

c) Metrics

Summarization models use **ROUGE metrics** instead of accuracy scores to verify how good the model is. The ROUGE scores used were **rouge1, rouge2, rougeL** and **rougeLsum.** It is preferred to use **rougeL** since it uses the **least common sequence** to produce its score. Since its **rougeL** score is 0.087, it is not a good model to use.

The **scores** for the **pre-trained** model are:

rouge1 average: 0.0873561269402484
rouge2 average: 0.0
rougeL average: 0.08728537224163033
rougeLsum average: 0.0873561269402484

```
{'rouge1': 0.05319148936170213, 'rouge2': 0.0, 'rougeL': 0.05319148936170213, 'rougeLsum': 0.05319148936170213}
{'rouge1': 0.0390625, 'rouge2': 0.0, 'rougeL': 0.0390625, 'rougeLsum': 0.0390625}
rouge1 average: 0.0873561269402484 - rouge2 average: 0.0 - rougeL average: 0.08728537224163033 - rougeLsum average: 0.0873561269402484
```

Figure 21: Rouge Metric Results

d) End Result:

```
category_dict_df = pd.DataFrame(columns = ['Product Category', 'Rating', 'Summary of reviews'])

for key in category_dict: # for each product category
    for rating in range(1, 6): # from 1 stars to 5 stars, rating
        category_dict_df.loc[len(category_dict_df)] = [key, rating, category_dict[key][rating - 1]]

display(category_dict_df)
```

✓ 0.0s

	Product Category	Rating	Summary of reviews
0	Computers,Electronics Features,Tablets,Electro...	1	the whitepaper looks Identical to the $120 mod...
1	Computers,Electronics Features,Tablets,Electro...	2	screen too dark The screen is too dark, and ca...
2	Computers,Electronics Features,Tablets,Electro...	3	NULL
3	Computers,Electronics Features,Tablets,Electro...	4	the kindle is good to download apps for books ...
4	Computers,Electronics Features,Tablets,Electro...	5	the amazon Kindle is light weight and easy to ...
...	...	...	...
110	Tablets,Fire Tablets,Electronics,iPad & Tablet...	1	very cheap and was not impressed at all never ...
111	Tablets,Fire Tablets,Electronics,iPad & Tablet...	2	NULL
112	Tablets,Fire Tablets,Electronics,iPad & Tablet...	3	the battery is having more and more trouble ho...
113	Tablets,Fire Tablets,Electronics,iPad & Tablet...	4	my daughter has had this tablet for almost 2 m...
114	Tablets,Fire Tablets,Electronics,iPad & Tablet...	5	NULL

115 rows × 3 columns

Figure 22: Results Example

2) Fine Tuning Model

The dataset used: **'gopalkalpande/bbc-news-summary'**
from **Dataset library**

Fine-Tuning

```python
dataset = load_dataset('gopalkalpande/bbc-news-summary', split = 'train')
full_dataset = dataset.train_test_split(test_size = 0.2, shuffle = True)

dataset_train = full_dataset['train'] # full_dataset['train'] # text?
dataset_valid = full_dataset['test'] # cambiar por category_dict[all_categ

print(dataset_train)
print(dataset_valid)
```

✓ 4.1s

```
Dataset({
    features: ['File_path', 'Articles', 'Summaries'],
    num_rows: 1779
})
Dataset({
    features: ['File_path', 'Articles', 'Summaries'],
    num_rows: 445
})
```

The model used: **t5-small** from https://huggingface.co/google-t5/t5-small

a) Tokenization

For the tokenization the T5 library provided an instruction to encode the data to be summarized.

b) Metrics

Rouge1	Rouge2	Rougel
0.898500	0.828500	0.881800

Figure 23: Rouge Results (Fine tune Model)

The **scores** for the **fine-tuned** model were:

rouge1: 0.0898500
rouge2: 0.828500
rougeL: 0.881800

It had **10x** a better **rougeL score** than the previous model. So this model would be a whole lot more usable for text summarizaton.

4) Transformer approach (HuggingFace API) for Sentiment

1) Foundation model (No Fine tunning)

-For this case, we use the BERT model. This model has already be pre-trained to classify products reviews into ratings.

-Dataset: 1429_1.csv

from https://www.kaggle.com/datasets/datafiniti/consumer-reviews-of-amazon-products

-Foundation Model: `bert-base-uncased`

a) Data Preporcessing:

- Clean nulls
- Drop unnecessary columns
- Remove short and super long reviews.
- Group together rating and Map them to Sentiment: Rating 1-2 = Negative, Rating 3 = neutral and rating 4-5 = positive.

b) Load Model and Tokenization

```
# Load models
print("Loading BERT sentiment model...")
tokenizer_sentiment = AutoTokenizer.from_pretrained('bert-base-uncased')
model_sentiment = AutoModelForSequenceClassification.from_pretrained('bert-base-uncased')
```

Figure 24: Model and Tokenizer

c) Classification Report:

```
Classification Report:
              precision    recall  f1-score   support

    negative      0.395     0.755     0.518       730
     neutral      0.288     0.419     0.341      1404
    positive      0.980     0.939     0.959     31372

    accuracy                          0.913     33506
   macro avg      0.554     0.704     0.606     33506
weighted avg      0.938     0.913     0.924     33506
```

Figure 25: Evaluation Results

Confusion Matrix:

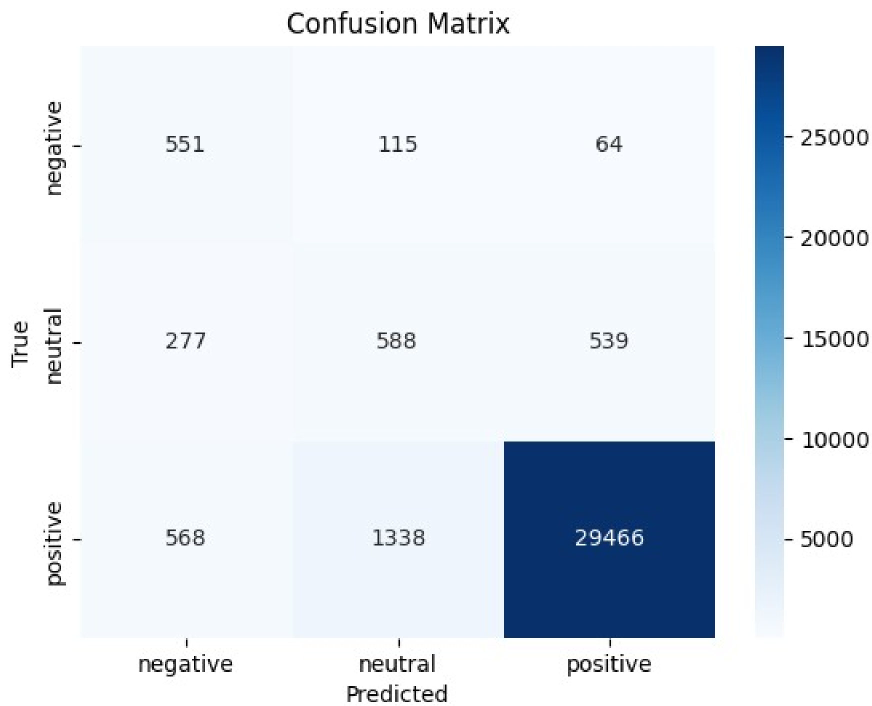

Figure 26: Confusion Matrix

2) Fine tuned Model (BERT)

-We decided to do full fine-tune after not improving the result with the LoRA Configuration. Since our dataset has 33,506 reviews and we have the computational resources.

a) Data Preporcessing:

-Similar data preprocessing as in the Foundation model.

How data distribution looks before grouping them into sentiment:

```
Distribución de ratings:
reviews.rating
1.0      367
2.0      363
3.0     1404
4.0     8200
5.0    23172
Name: count, dtype: int64

Distribución de sentimientos:
sentiment
0      730
1     1404
2    31372
Name: count, dtype: int64
```

Figure 27: Data Distribution

b) Split data into Training and Validation

```
# Dividir los datos
train_texts, val_texts, train_labels, val_labels = train_test_split(
    df['reviews.text'].tolist(),
    df['sentiment'].tolist(),
    test_size=0.2,
    random_state=42
)
```

c) Load Model and Tokenizer:

```
# Cargar el tokenizer y modelo BERT
tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased')
model = AutoModelForSequenceClassification.from_pretrained(
    'bert-base-uncased',
    num_labels=3
)
```

Figure 28: Tokenizer and Model

d) Configure dataloader and Train model:

```
# Configurar dataloaders con un tamaño de batch más pequeño
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32)

# Entrenamiento
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
optimizer = torch.optim.AdamW(model.parameters(), lr=2e-5)
```

e) Evaluation using the full dataset:

```
==================================================
Full Dataset Evaluation Results:
==================================================
Average Loss: 0.0726
Accuracy: 0.9835
F1 Score (weighted): 0.9831

Detailed Classification Report:
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.90      0.88      0.89       730
           1       0.86      0.78      0.82      1404
           2       0.99      1.00      0.99     31372

    accuracy                           0.98     33506
   macro avg       0.92      0.88      0.90     33506
weighted avg       0.98      0.98      0.98     33506
```

Figure 29: Evaluation Results

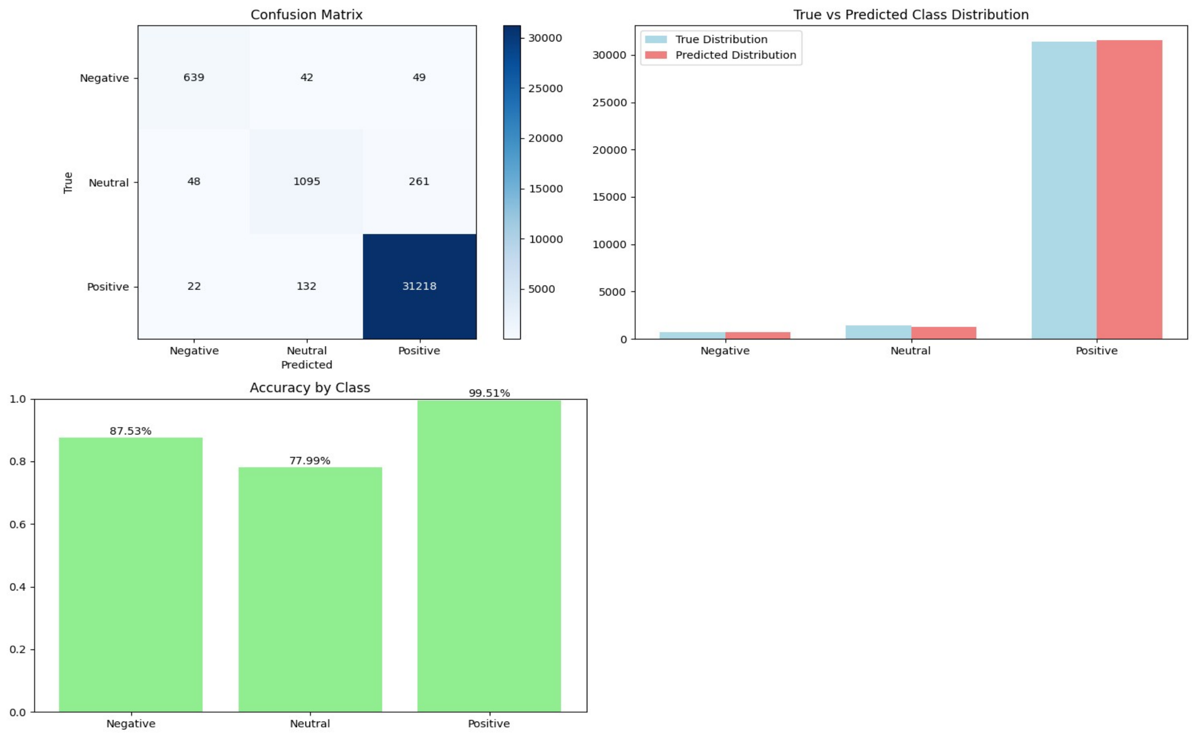

Figure 30: Evaluation Plots

BERT Sentiment Analysis Model Performance Analysis

Overall Performance
- Accuracy: 97.95%
- F1 Score (weighted): 97.87%
- Loss: 0.0755

Class-wise Performance
1. Positive Sentiment (Class 2)
   - Precision: 99%
   - Recall: 100%
   - F1-score: 99%
- Best performing class due to majority representation

2. Neutral Sentiment (Class 1)
   - Precision: 82%
   - Recall: 70%
   - F1-score: 76%
   - Moderate performance, struggles with recall

3. Negative Sentiment (Class 0)
   - Precision: 87%
   - Recall: 84%
   - F1-score: 86%
   - Good performance despite small representation

Key Observations
1.Class Imbalance Impact
   - Model performs exceptionally well on majority class (Positive)
   - Maintains good performance on minority classes despite imbalance

2. Confusion Matrix Analysis
```
[[  615    76    39]
 [   71   985   348]
 [   17   136 31219]]
```
   - Most misclassifications occur between neutral and positive classes
   - Very few negative reviews misclassified as positive (39 cases)

3. Training Behavior
   - Model converged well with decreasing loss
   - No significant overfitting observed
- Consistent performance across training epochs

Recommendations
1. Consider techniques to address class imbalance:
   - Oversampling minority classes
   - Class weights in loss function
   - Collecting more negative and neutral reviews

2. Potential improvements:
   - Longer training with early stopping
   - Data augmentation for minority classes
   - Fine-tuning hyperparameters

Overall, the model shows strong performance despite the significant class imbalance, making it suitable for production use in sentiment analysis tasks.