Project NLP |

Business Case:

Automated Customer Reviews

Abdulaziz Alshehri   Bader Albehishi   Amjad Aloufi

Table of Contents

Introduction

This report presents an NLP-based solution to automate the analysis of customer product reviews. With the increasing volume of online reviews, manual processing is no longer practical. The proposed system leverages modern language models to classify sentiments, cluster products, and generate summaries for user-friendly recommendations.

Background or Context

E-commerce platforms contain numerous product reviews that offer valuable insights. Manually extracting and organizing this information is inefficient. Recent advancements in natural language processing and transformer-based models provide practical tools to automate and enhance this process. Practical tools to automate and improve this process.

Purpose of the Report

The purpose of this report is to outline the development and evaluation of a system based on a given dataset (Amazon customer reviews) that:

- Classifies reviews into sentiment categories (positive, neutral, negative).
- Cluster products into broader categories.
- Summarize key insights using generative AI models.
- Deploy (Classification, Clustering, and Summarization models) to be accessible as an end-user-friendly interface.

1. Review Classification

Objective: To classify customer reviews as positive, negative, or neutral to support product and service improvements.

Task: To develop a model that analyses the textual content of customer reviews and accurately assigns them to one of the three sentiment categories.

Dataset Description: The dataset is loaded from a CSV file, retains only the relevant columns from the file (reviews.text and reviews. rating). The dataset contains 10,098 reviews with ratings from 1.0 to 5.0. Grouped into sentiment classes: Positive (4–5) – 4,686 reviews, Neutral (3) – 2,902 reviews, and Negative (1–2) – 2,510 reviews. The data is imbalanced, favouring positive sentiment.

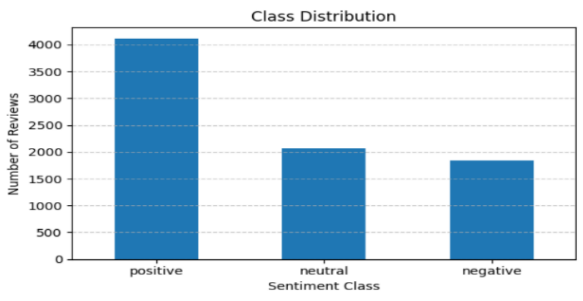
Model Development:

1. **Importing Libraries**
   The code begins by importing necessary libraries for data processing (pandas, numpy), model development and tokenization (transformers, torch), and evaluation (sklearn). These enable smooth execution of the sentiment classification pipeline.

2. **Loading the Dataset**
   Two CSV files are loaded: one for training (train.csv) and one for validation (val.csv). The dataset contains customer reviews and their corresponding sentiment labels. These files were previously cleaned and prepared.

3. **Label Encoding**
   Sentiment classes ("positive", "neutral", "negative") are converted into numerical labels using LabelEncoder, which is essential for compatibility with the model's classification layer.

4. **Handling Class Imbalance**
   The dataset is highly imbalanced, with a dominance of the **positive class**. To address this, the code computes **class weights** using compute_class_weight. These weights are then used to give more importance to underrepresented classes (neutral and negative) during training.



5. **Tokenizing Review Texts**
   The textual data is tokenized using RobertaTokenizerFast from the Hugging Face library. The tokenizer converts raw review texts into token IDs while ensuring consistent input lengths using padding and truncation.

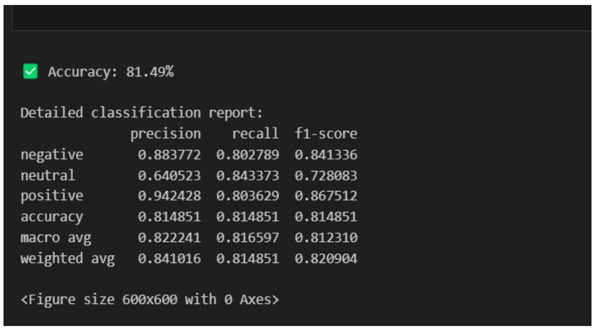
6. **Preparing Dataset Objects**
   A custom PyTorch Dataset class wraps the tokenized inputs and encoded labels. This makes it compatible with Hugging Face's Trainer API and supports efficient batch training.

7. **Initializing the Pretrained Model**
   The model used is roberta-base, a powerful transformer architecture pretrained on a large English corpus. It is loaded using RobertaForSequenceClassification with num_labels=3 to handle the three sentiment categories. The classifier's weight layer is adjusted using the computed class weights to prioritize underrepresented classes during training.

8. **Defining Training Arguments**
   Training hyperparameters are configured using TrainingArguments. These include a learning rate of 2e-5, batch size of 16, number of epochs set to 4, and an evaluation strategy that runs at the end of every epoch. The best model is saved based on validation loss.

9. **Model Training via Trainer**
   The Hugging Face Trainer class is used to manage the training process. It handles training loops, evaluation, checkpointing, and integration of class weights, streamlining the fine-tuning process.

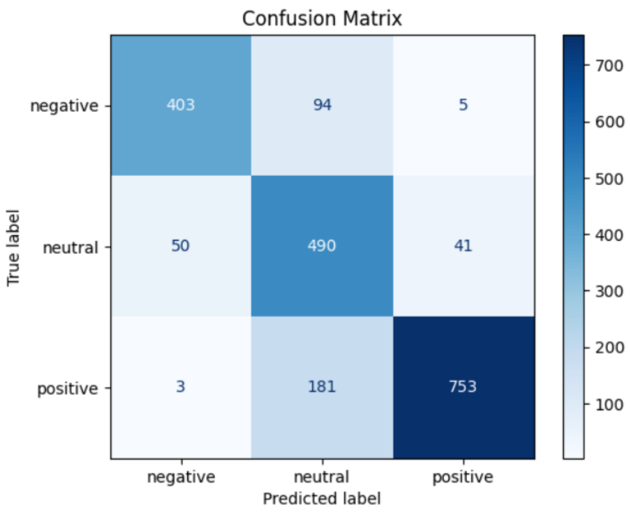
10. **Evaluation and Performance Reporting**
    After training, the model is evaluated on the validation set. Predictions are generated and compared against true labels using metrics such as **precision**, **recall**, and **F1-score**. This step helps verify that the class imbalance was handled effectively and that all sentiment categories are being learned.



```
✓ Accuracy: 81.49%

Detailed classification report:
              precision    recall  f1-score
negative       0.883772  0.802789  0.841336
neutral        0.640523  0.843373  0.728083
positive       0.942428  0.803629  0.867512
accuracy       0.814851  0.814851  0.814851
macro avg      0.822241  0.816597  0.812310
weighted avg   0.841016  0.814851  0.820904

<Figure size 600x600 with 0 Axes>
```



Previous attempts:

- In this version of the classification model, the issue of class imbalance was addressed using an oversampling strategy, wherein the minority sentiment classes—neutral and negative—were replicated in the training set to equal the number of positive examples. This method enabled the use of the standard cross-entropy loss function without requiring class weighting, as the training distribution was manually balanced. Although oversampling contributed to improved fairness and more consistent performance across the sentiment categories, the model continued to underperform on the neutral class. This outcome suggests that oversampling alone may be insufficient to capture the inherent ambiguity or subtlety often associated with neutral sentiment.

Limitations and Challenges:

- **Severe Class Imbalance in Available Data**
  Most datasets used in this project were significantly imbalanced, with a high number of positive

reviews compared to neutral and negative ones. This imbalance made it challenging for the model to learn to represent all classes equally.

- **Difficulty Applying Emotion-Based Models**
  Emotion classification models could not be fully utilized due to the lack of labeled emotion data. Without corresponding emotion labels, it was not possible to train or evaluate models based on emotional categories.
- **Mismatch Between Review Text and Star Ratings**
  There was a frequent inconsistency between the tone of the review and its assigned star rating. For example, in many instances, reviews appeared positive in tone but were rated poorly (e.g., 1 or 2 stars). This discrepancy introduced noise into the labeling process and made it difficult to rely on star ratings as a definitive measure of sentiment.
- **Limited Multi-Class Sentiment Labels**
  The majority of datasets contained only binary sentiment labels (positive and negative), with very limited examples labeled as neutral. This limited the development of a robust three-class sentiment classification system.

2. Product Category Clustering

Objective: To Group product reviews into 4–6 high-level meta-categories to simplify analysis and improve insight extraction.

Task: Develop a clustering model to categorize all product reviews into broader groups. (Exe:Ebook Readers, Batteries, ...).

Dataset Description: The dataset contains 34,660 customer reviews across a wide range of products. It includes 21 columns covering product metadata, review content, user feedback, and timestamps. After multiple experiments, the final set of columns used in the analysis included only those deemed most relevant to the task.

Categories: Defines the product hierarchy (e.g., "Electronics, Tablets"). Used to group reviews by main category.

Name: Contains the product names (available in 27,900 rows). Used to identify and summarize individual products.

Model Development

1. Install Required Libraries

Installs packages including matplotlib, wordcloud, sentence_transformers, and hdbscan.These libraries are essential for text visualization, sentence embedding, and unsupervised clustering.

2.Import Dependencies

Loads libraries for text processing, embedding, clustering algorithms, evaluation metrics, and visualization. Prepares the environment with tools needed for clustering and interpretation.

3.Download NLTK Resources

Downloads wordnet and stopwords from NLTK. Enables lemmatization and filtering of common stopwords during text preprocessing.

4.Load Dataset

Loads a CSV file containing product review data. Provides the input data (text reviews) needed for embedding and clustering.

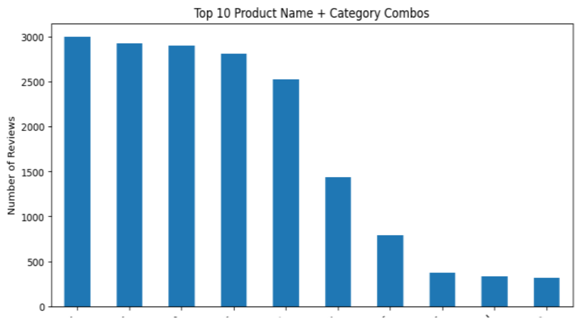
5.Text Preprocessing (in later cells)

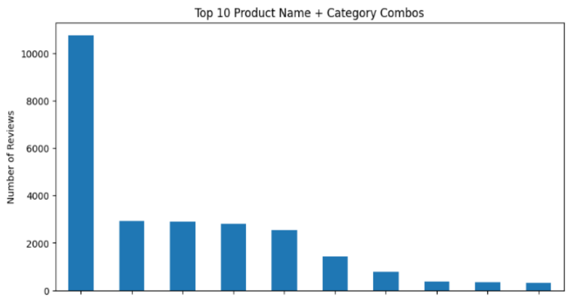
Applies tokenization, stopword removal, punctuation stripping, and lemmatization. Cleans the review text to improve embedding quality and cluster coherence.

6.Balancing Dataset

Undersampling the dataset to prevent products with too many reviews from dominating the dataset and that helps produce more balanced clusters.

Before balancing the dataset





7.Generate Sentence Embeddings

Uses a ‘all-MiniLM-L6-v2’ SentenceTransformer model to convert cleaned reviews into vector embeddings. Transforms text into a numerical format suitable for clustering.

8.Dimensionality Reduction with UMAP

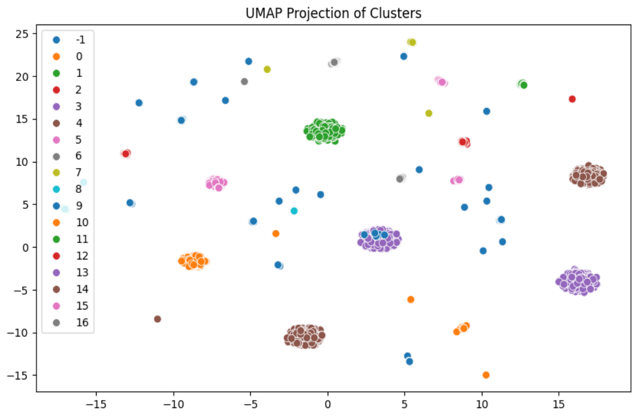
Applies UMAP to reduce high-dimensional embeddings for visualization. Makes it easier to visualize and understand clustering structure in 2D space.

9.Apply HDBSCAN Clustering

Performs density-based clustering using HDBSCAN on the embeddings. Identifies natural groupings in the data without requiring a fixed number of clusters.

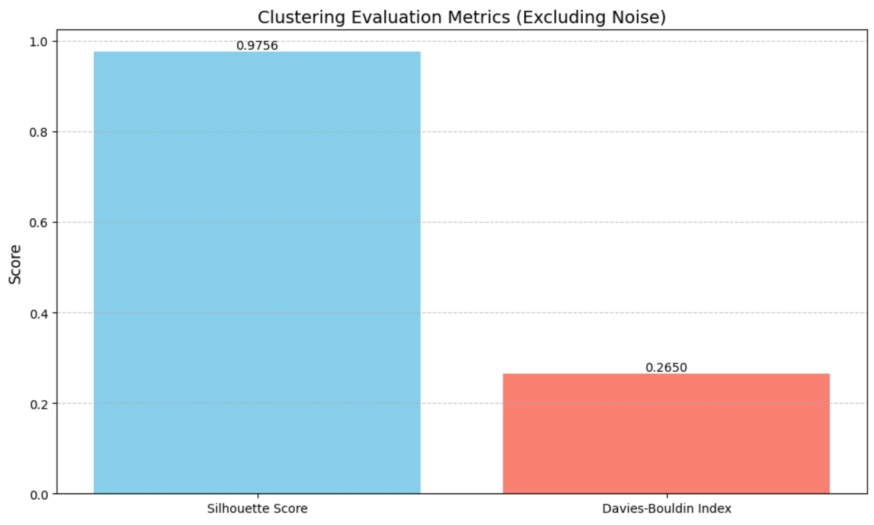
10.Visualize the cluster using UMAP

This UMAP plot shows the clustered embeddings in 2D space, with each color representing a distinct group. Points labeled -1 are considered noise and not assigned to any cluster.



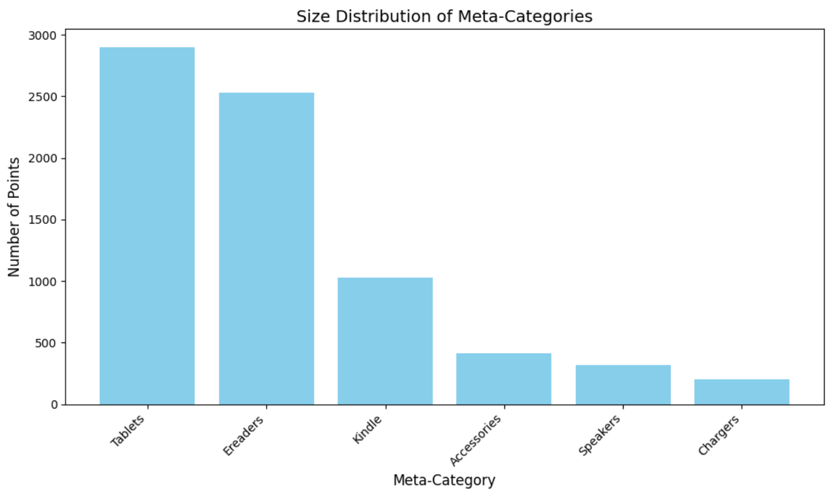
11.Evaluate Clustering Quality

Uses metrics like Silhouette Score, Davies-Bouldin Index, and Calinski-Harabasz Score. Quantifies how well the clusters represent distinct groupings.



12.Meta-Category Generation and Assignment using GPT and Clustering

Automate meta-category generation by analyzing clustered text data and grouping them by using GPT prompting to assign meaningful category names based on sample items in each cluster.



Previous attempts:

- Applied unsupervised clustering for product meta-categorization using SentenceTransformer and K-Means. Used only a categories-cleaned text field and there was miss subtle differences between products with the same category tag but different use cases.. Text embeddings are generated using the paraphrase-MiniLM-L3-v2 model from Hugging Face’s SentenceTransformer and its was not that efficient for capturing semantic differences between product names and categories. K-Means is applied to the embeddings to form six clusters but it proved insufficient for capturing the deeper, more nuanced structure of the dataset, representing broad product groupings. Meta-category names are manually assigned to each cluster and that was not automated.

- Tried different HDBSCAN min_cluster_size parameter values to attempt the best cluster.

Limitations and Challenges:

Deciding which cluster method or feature to use in clustering was a challenge, so applying and trying different methods helped in choosing the appropriate cluster.

Feature/Cluster method	K-means	HDBSCAN
Name	Silhouette score: 0.11 Davies-Bouldin: 2.35	Silhouette score: 0.9172 Davies-Bouldin: 0.3482
Categories	Silhouette score: 0.12 Davies-Bouldin: 1.97	Silhouette score: 0.9483 Davies-Bouldin: 0.2942
Combine name+categories	Silhouette score: 0.30 Davies-Bouldin: 1.48	Silhouette score: 0.9756 Davies-Bouldin: 0.2650

3. Review Summarization Using Generative AI

Objective: To create article-style summaries that provide recommendations for the top products in various categories based on customer reviews.

Task: To develop a model that generates succinct narratives for each product category, with an emphasis on:

- Identifying the top three products and highlighting their distinguishing features
- Presenting key customer complaints related to each of the top products
- Identifying the least favourable product and outlining the reasons for its avoidance

Model Development:

1. **Receive the Input Category**
   The system begins by accepting a product category through the /summarize API endpoint. This allows users to request insights for any specific product group dynamically.
2. **Validate Data Availability**
   It checks whether the global review dataset (global_df) is loaded and accessible. Summarization cannot proceed without access to review data.
3. **Filter the Dataset by Category**
   The data is filtered to include only entries matching the requested category (case-insensitive). Ensures that the summary focuses solely on relevant products within the selected group.
4. **Ensure Required Columns Are Present**
   The system verifies the presence of reviews.rating, name, and reviews.text columns. These fields are essential for identifying product performance and extracting review content.
5. **Check Category Validity**
   If no matching records are found, a 404 error is returned. Prevents generating irrelevant or misleading summaries.
6. **Compute Average Ratings Per Product**
   For all products in the category, the average of reviews.rating is calculated. This quantifies product performance based on customer feedback.

7. **Ensure Minimum Product Count**
The function ensures there are at least four products to compare. A meaningful summary requires contrast between multiple high- and low-performing products.

8. **Identify Top and Bottom Products**
The top three products (highest average ratings) and the lowest-rated product are selected. These will serve as the highlights and warning points in the narrative summary.

9. **Extract Customer Review Snippets**
For each selected product, up to five textual reviews are extracted. These snippets provide authentic language and sentiment for the model to draw from.

10. **Construct the Prompt for GPT**
A carefully worded prompt is generated, including review snippets, product names, and narrative instructions. A well-structured prompt guides the language model to write a consistent, engaging, and sentiment-rich article.

11. **Generate the Summary**
The prompt is passed to a language model (e.g., GPT) to produce a ~200-word summary article. This provides users with a concise, narrative-style overview of the category based on real user opinions.

12. **Return the Final Output**
The API returns a JSON object with the category name, the generated summary, top three product names, and their average ratings. This structured output can be used in dashboards, reports, or recommendation systems.

Previous attempts:

- Applied zero-shot summarization approach using the BART transformer model. It begins by preprocessing customer review data, cleaning the review text and extracting primary product categories. The pipeline from Hugging Face Transformers is used to generate abstractive summaries of reviews without prior task-specific training. The summarization is applied across different product categories. The outputs generated by the zero-shot model were too brief and did not fulfil the project's requirements.
- Applied few-shot prompting for abstractive summarization using the google/flan-t5-base model. It begins by cleaning textual review data and then utilizes Hugging Face's pipeline API to generate summaries. Example-based prompting is employed to guide the model in producing human-like, contextual summaries based on a few review samples. The few-shot model was applied successfully; however, in this case, it did not produce satisfactory results.

Deployment

Application Deployment Overview:

This application is built to analyse product reviews using machine learning. It can detect the sentiment (positive, neutral, negative), group similar reviews, and generate summaries using artificial intelligence. The backend system is developed using FastAPI, a modern Python framework for creating web applications.

1. Starting the Application

The system runs as a web service using FastAPI. It sets up folders for HTML files and static content (like images or CSS), and creates the main application that users and other systems can interact with through a browser or API.

2. Sentiment Classification

A trained machine learning model (based on BERT) is loaded to classify review text. When a user submits a review, the model predicts whether it is positive, neutral, or negative.

- Example: A review like "I love this product!" would be labeled as Positive.
- This function is available through an API endpoint called /classify.

3. Generating Summaries

The app uses OpenAI’s GPT model to write a short summary of customer opinions for a product category. It reads real reviews and writes an article that highlights the top-rated products and mentions customer concerns.

- This feature is useful for creating readable summaries from many reviews.
- It's triggered through the /summarize endpoint.

4. Grouping Reviews by Topic (Clustering)

Instead of reading every review one by one, the app groups similar reviews into categories using a method called clustering. It uses a model to turn each review into a numeric format (embedding), then groups similar ones together.

- This helps in understanding common themes or issues in product feedback.
- The app can also assign names to each group using GPT (like "Tablets" or "Smart Devices").

5. Uploading Review Files

Users can upload a .csv file that contains reviews, product names, and categories. The app checks if the required columns are there and then stores the data in memory so other parts of the app can use it.

- The upload is handled by the /upload_csv endpoint.

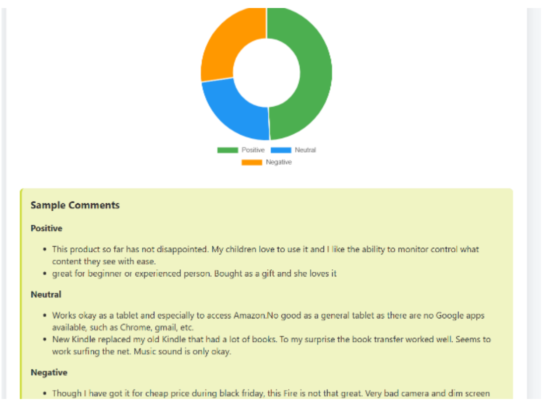
6. Additional Features

The app also includes:

- A simple method to estimate how many reviews are positive, neutral, or negative using keyword rules.
- A function that shows which product categories appear most often in the uploaded file.

7. Web Interface

The app includes a simple web page (HTML) where users can upload files, type in reviews, and view results. This makes it easier to use for people who don't want to use the API directly.



Sample Comments

Positive

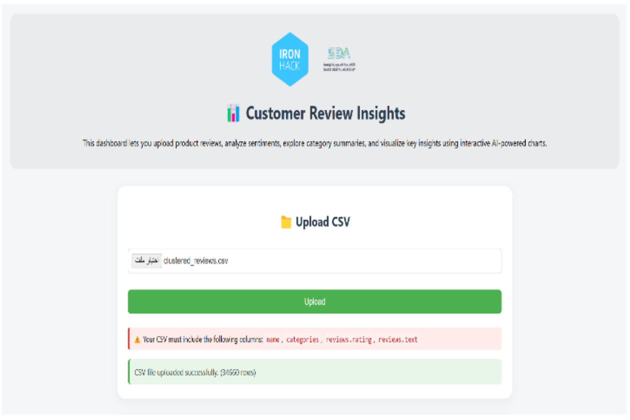
- This product so far has not disappointed. My children love to use it and I like the ability to monitor control what content they see with ease.
- great for beginner or experienced person. Bought as a gift and she loves it

Neutral

- Works okay as a tablet and especially to access Amazon.No good as a general tablet as there are no Google apps available, such as Chrome, gmail, etc.
- New Kindle replaced my old Kindle that had a lot of books. To my surprise the book transfer worked well. Seems to work surfing the net. Music sound is only okay.

Negative

- Though I have got it for cheap price during black friday, this Fire is not that great. Very bad camera and dim screen



Customer Review Insights

This dashboard lets you upload product reviews, analyze sentiments, explore category summaries, and visualize key insights using interactive AI-powered charts.

Upload CSV

⚠ Your CSV must include the following columns: `name`, `categories`, `reviews.rating`, `reviews.text`

CSV file uploaded successfully. (34660 rows)

8. Security and Setup

The system uses a secret API key to access OpenAI's GPT service. This key is read securely from your computer's environment settings.

9. How It Runs

The app runs locally on your machine using uvicorn, a server that powers FastAPI apps. You can access it by visiting http://127.0.0.1:8000 in your web browser.

Limitations and Challenges:

- **Clustering-to-Summarization Link**
  Ensuring smooth data flow from clustering to summarization posed a challenge, as it required extracting key product insights from each cluster and formatting them effectively for the GPT-based summarization model.
- **Model Pipeline Coordination**
  While not overly complex, integrating the three models—classification, clustering, and summarization—into a single pipeline required attention to data flow, compatibility, and deployment consistency.

Conclusion

This project presented a complete NLP pipeline for automating customer review analysis. By combining sentiment classification, product clustering, and AI-generated summarization, the system transforms unstructured review data into clear, actionable insights. The deployed application offers a user-friendly interface and demonstrates the practical value of integrating multiple models in a cohesive, real-world solution.