Final Report

Introduction:

This project focused on leveraging machine learning techniques to extract deeper insights from the Amazon Fine Food Reviews dataset. The goal was to design systems that could not only predict customer review ratings, but also recommend similar products and identify helpful reviews. Using a combination of traditional machine learning models, natural language processing (NLP) techniques, and structured workflows within AWS SageMaker, we explored practical challenges like data imbalance, high dimensionality, and scalability. Each part of the project targeted a real-world application of text analytics to improve customer experience and decision-making processes in e-commerce platforms.

Objective:

The primary objectives of this project were threefold: (1) to build a content-based recommendation system that identifies similar products based on customer review text, (2) to develop a robust machine learning classification system capable of predicting review ratings while handling data imbalance and large feature spaces, and (3) to predict review helpfulness scores and select the most representative reviews for products.

Part -1: Content-Based Recommendation System for Amazon Reviews

1. Methodology

This part of our project aimed to develop a content-based product recommendation system using Amazon review data. The core machine learning approach involved leveraging natural language processing (NLP) techniques to represent products based on the semantic content of their reviews and recommending items with similar content.

- Algorithm Selection:
  - Embedding Generation: The sentence-transformers library (specifically the pre-trained all-MiniLM-L6-v2 model) was selected to generate dense vector representations (embeddings) for each review's cleaned text (CleanedText). This model was chosen for its balance of performance and computational efficiency suitable for running within a notebook environment.
  - Product Representation: A single embedding vector for each unique ProductId was created by averaging the embeddings of all associated reviews.
  - Similarity Calculation: Cosine similarity was used to measure the similarity between the averaged product embedding vectors.
  - Recommendation: For a given target product, other products were ranked in descending order of their cosine similarity score, providing a list of content-based recommendations.
- Data Splitting:

    - No explicit train/test split was performed for the evaluation of the recommendation task itself, as the primary evaluation method was qualitative and the system uses all available text to build product profiles. The preprocessing steps were applied to the entire dataset used for building the recommender.
- Hyperparameter Tuning:
    - No hyperparameter tuning was performed for the selected pre-trained sentence-transformers model or the cosine similarity calculation in this implementation.
- Feature Selection:
    - The core features used for the recommendation logic were ProductId (to identify items) and CleanedText (the preprocessed review text used for embedding generation).

2. Building Machine Learning Pipelines in AWS SageMaker

Content-Based Recommendation System for Amazon Reviews -

The automation of the ML workflow using AWS SageMaker Pipelines was not implemented in this part of the project because we did not have the necessary IAM Permission. The entire process, including data loading from S3, preprocessing, feature engineering, embedding generation (using sentence-transformers), similarity calculation, and recommendation generation, was executed sequentially within a single Jupyter Notebook (manually_balanced_eda-2.ipynb) hosted on an AWS SageMaker notebook instance.

- Workflow:
    - Data flowed linearly through the notebook cells: loading -> cleaning -> embedding generation -> similarity matrix calculation -> recommendation function definition -> qualitative evaluation.
- SageMaker Pipelines Potential:
    - While not used here, AWS SageMaker Pipelines provide a robust mechanism for automating, orchestrating, and managing ML workflows. In future work, this workflow could be refactored into distinct steps (e.g., Preprocessing, Embedding Generation, Evaluation) within a SageMaker Pipeline for improved reproducibility, scalability, and integration with CI/CD practices.

3. Model Training in AWS SageMaker

Content-Based Recommendation System for Amazon Reviews -

Training of models using AWS SageMaker built-in algorithms or dedicated training jobs was not performed in this part because we did not have the necessary IAM Permission. This decision was influenced by the code's focus on leveraging pre-existing text embeddings and potential constraints associated with the LabRole IAM permissions.

- Approach Used:
    - Instead of training, a pre-trained NLP model (all-MiniLM-L6-v2 via sentence-transformers) was loaded directly within the notebook environment and used for inference to generate text embeddings. The "model" in this context is the embedding generation process followed by similarity calculation.
- Hyperparameter Tuning/Optimization:

    - As no model training occurred, no hyperparameter tuning or optimization specific to a SageMaker training job was conducted.
- Comparative Results:
    - Only one embedding model (all-MiniLM-L6-v2) was used; therefore, no comparison between different models trained via SageMaker was performed.
- SageMaker Training Potential:
    - For tasks requiring custom model training or fine-tuning (e.g., fine-tuning an embedding model on the review data, training a classification model), AWS SageMaker provides managed training capabilities that allow users to run training jobs using built-in algorithms or custom scripts within dedicated, scalable environments. This was outside the scope of the current part of the project.

4. Performance Evaluation Using AWS SageMaker Tools

Content-Based Recommendation System for Amazon Reviews -

Direct evaluation using standard classification/regression metrics (accuracy, precision, recall, F1-score, RMSE) was not appropriate or feasible for this specific part of our project where we do recommendation system implementation due to the lack of explicit ground truth (i.e., pre-defined relevance labels for product pairs). Consequently, AWS SageMaker's built-in model monitoring or evaluation tools, which often rely on such metrics, were not utilized because of the limited permissions in the IAM role.

- Evaluation Method Used:
  The primary evaluation method employed was Qualitative Spot-Checking, as recommended for sanity-checking recommender systems.
  This involved:
    - Manually inspecting the ProductIds of both the target and recommended items. This involved cross-referencing these IDs, potentially using the original Reviews.csv data about the products, to assess relevance.
    - Subjectively judging whether the recommended products were logically related to the target product based on likely content similarity (e.g., same category, brand, flavor, intended use).
- Performance Assessment:
    - Based on this manual qualitative spot-checking process, the recommendation system demonstrated reasonable performance. For the products, most of the top 5 recommendations appeared to be relevant and contextually appropriate to the target product. In some instances, one out of the 5 recommendations seemed ambiguous or less clearly related, which might be attributed to factors like generic language in reviews or the averaging of diverse review embeddings. Overall, the qualitative assessment suggests the content-based approach successfully captured meaningful similarities between products based on their review text.

6. Results and Discussion

Content-Based Recommendation System for Amazon Reviews -

- Key Findings:
    - The qualitative evaluation indicated that the content-based recommendation system, using averaged sentence-transformers embeddings, was capable of identifying products with apparent textual similarity. In many spot-checked cases, the recommended items belonged to similar categories or likely shared common attributes discussed in reviews, as suggested by their high cosine similarity scores. However, instances of less intuitive recommendations were also observed, potentially due to generic terms dominating reviews or limitations in capturing nuanced product differences solely from averaged review text.
- Strengths:
    - Leverages the rich textual content of user reviews.
    - Simple to implement once embeddings are generated.
    - Does not require user interaction data (avoids the user cold-start problem).
- Limitations:
    - Content-Only Focus: Ignores user preferences, purchase history, or collaborative patterns. Recommendations can feel narrow or obvious.
    - Item Cold-Start: New products with few or no reviews cannot be effectively recommended or used as targets.
    - Embedding Averaging: Averaging review embeddings might dilute specific details or conflicting sentiments within the reviews for a single product.
    - Scalability: Generating embeddings and computing the full similarity matrix within a notebook can be memory and computationally intensive for very large datasets.
- Challenges Encountered:
    - Environment Constraints: The LabRole limited the use of integrated SageMaker services, necessitating a notebook-centric approach with external libraries.
    - Execution Order: NameError issues highlighted the sensitivity of Jupyter Notebooks to cell execution order, requiring careful sequential execution.
    - Memory Management: Generating embeddings for hundreds of thousands of reviews required attention to memory usage.
- Code Structure:
    - The code follows a sequential flow typical of notebooks, progressing from data loading and preprocessing to embedding generation, similarity calculation, and evaluation. Functions were defined for text cleaning (clean_text, expand_contractions), recommendation (get_recommendations), and evaluation (display_sample_recommendations_eval).
- Documentation:
    - Markdown cells were used to explain the major steps, particularly during the EDA and preprocessing phases. Code comments were present but could be enhanced in the recommendation and evaluation sections for better clarity.
- Reproducibility:
    - While the notebook captures the steps, reproducibility relies heavily on executing cells in the correct order. Encapsulating core logic (like embedding generation or recommendation) into separate Python scripts or using a workflow orchestrator like SageMaker Pipelines would significantly improve reproducibility and modularity.
- Libraries Used:
    - Standard libraries like pandas, numpy, nltk, re, sklearn, and sentence-transformers were employed.

8. Conclusion and Future Work

Content-Based Recommendation System for Amazon Reviews -

This part of the project successfully implemented a functional content-based product recommendation system for Amazon reviews within a Jupyter Notebook environment. By leveraging pre-trained sentence embeddings and cosine similarity, the system demonstrated its ability to retrieve textually similar products, as confirmed by qualitative evaluation. The idea navigated environmental constraints by using the sentence-transformers library effectively.

- Future Work & Enhancements:
  - Hybrid Approaches: Combine the content-based system with collaborative filtering techniques (using user-item interaction data like ratings or purchase history) to create more personalized and robust recommendations.
  - SageMaker Integration: Refactor the workflow using AWS SageMaker Pipelines for automation, orchestration, better versioning, and reproducibility.
  - Scalable Training/Embedding: Utilize SageMaker Training Jobs if fine-tuning embedding models or training other components becomes necessary. For large-scale embedding generation, SageMaker Batch Transform could be considered.
  - Deployment: Deploy the recommendation model (specifically, the ability to generate embeddings and perform similarity lookups) using AWS SageMaker Endpoints for real-time inference.
  - Improved Evaluation: If user interaction data or relevance labels become available, implement standard recommendation metrics (Precision@k, Recall@k, MAP, nDCG).
  - Advanced Embeddings: Experiment with different or larger pre-trained embedding models, or fine-tune models specifically on the Amazon review dataset.
  - User Interface: Develop a simple interface (e.g., using Streamlit, Flask, or SageMaker endpoint integration) to interact with the recommendation system.

Part 2: Automated Rating Prediction from Food Reviews using Machine Learning

This part of our project focuses on building an ML-based classification system to predict customer review ratings from Amazon Fine Food Reviews, addressing challenges of data imbalance, feature extraction, and scalable model deployment.

Step 1: Data Upload and Retrieval

- A new S3 bucket (reviews-raw-data) was created using the boto3 library.
- The raw Amazon Fine Food Reviews dataset was downloaded, extracted, and uploaded to this S3 bucket.
- Data was retrieved from S3 into the SageMaker environment for preprocessing and analysis.

Step 2: Preprocessing and EDA

Preprocessing Steps

- **EDA**:
  - Missing values in the ProfileName column were imputed with 'Unavailable' to maintain data consistency, and the Summary column was dropped to streamline the dataset by removing redundant information.
  - **Duplicates** were identified and removed to avoid bias.
  - To prepare the text data, a custom cleaning function was applied that expanded common contractions, removed non-alphabetic characters, and lowercased the text.
  - Stopwords were removed and lemmatization was performed to standardize words, resulting in a new CleanedText feature for modeling.
- **Feature Engineering**:
  - Helpfulness Ratio = HelpfulnessNumerator / HelpfulnessDenominator
  - Sentiment Score and Subjectivity scores extracted using TextBlob
  - Text Length feature created by counting the number of words
- **Class Imbalance Handling**:
  - Observed imbalance in review scores.
  - A two-stage balancing strategy was implemented: first through **random undersampling** across review scores, and later through **class-weighted learning** during model training to address imbalance in the relabeled classes.

Step 3: Data Flow

- A structured **manual workflow** was followed from S3 data retrieval → preprocessing → model training → evaluation.

Step 4: Models Trained

Model 1: Support Vector Classifier Model

Feature Engineering

- To prepare the dataset for modeling, the textual data (CleanedText) was vectorized using TF-IDF (Term Frequency-Inverse Document Frequency), limiting to the top 5000 features to reduce noise and dimensionality.
- Alongside TF-IDF text features, numerical features such as Sentiment, Helpfulness Ratio, Text Length, and Subjectivity were combined to enrich the input space.

Dimensionality Reduction

- Given the high dimensionality of the TF-IDF feature space, Truncated Singular Value Decomposition (SVD) was applied to this part of the project the feature matrix onto a lower-dimensional subspace.
- The number of components was selected to retain over 95% variance, ensuring minimal information loss while significantly improving computational efficiency.

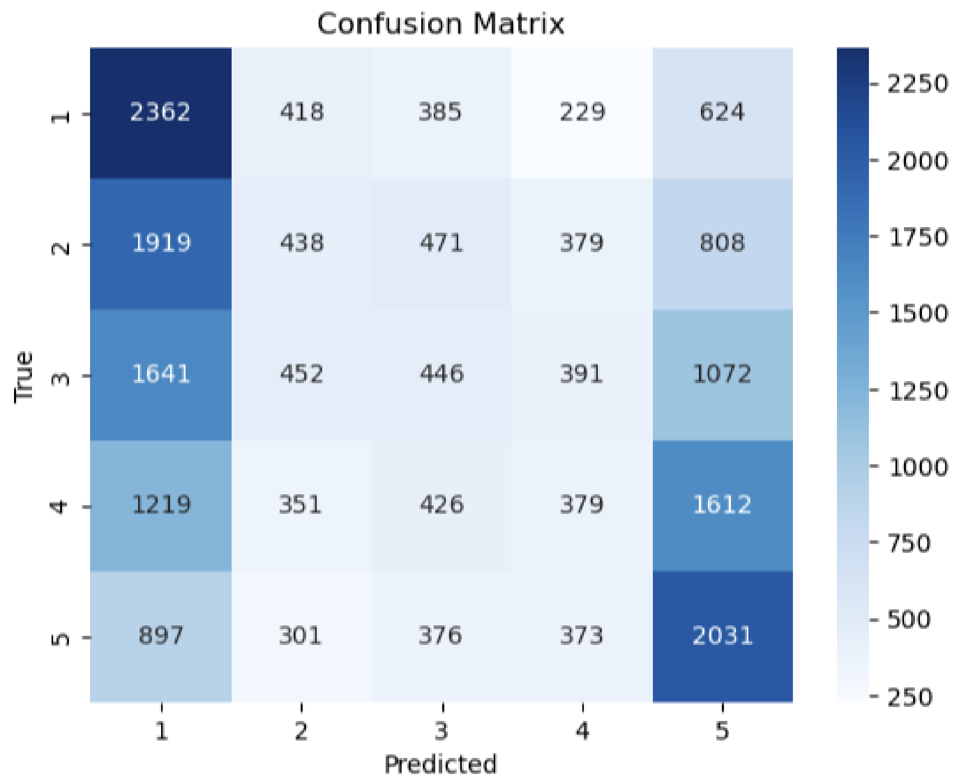
Model Training and Hyperparameter Tuning

- A Support Vector Classifier (SVC) with a linear kernel was trained.
- Hyperparameter tuning was performed using GridSearchCV, optimizing the C and gamma parameters through a 3-fold cross-validation process to select the best configuration.

Model Evaluation

- The best SVC model achieved an accuracy of approximately 28% on the test data.
- The confusion matrix and classification report highlighted challenges in correctly classifying minority classes, consistent with the dataset's complexity and imbalanced nature.

Support Vector Machines, even when operating on high-dimensional sparse text data, can serve as a baseline classifier for sentiment analysis tasks. Although performance was limited in this case, SVMs remain valuable for quick prototyping and benchmarking before deploying more complex models like Random Forests or XGBoost.



```
Confusion Matrix:
[[2362  418  385  229  624]
 [1919  438  471  379  808]
 [1641  452  446  391 1072]
 [1219  351  426  379 1612]
 [ 897  301  376  373 2031]]
Classification Report:
              precision    recall  f1-score   support

           1       0.29      0.59      0.39      4018
           2       0.22      0.11      0.15      4015
           3       0.21      0.11      0.15      4002
           4       0.22      0.10      0.13      3987
           5       0.33      0.51      0.40      3978

    accuracy                           0.28     20000
   macro avg       0.26      0.28      0.24     20000
weighted avg       0.26      0.28      0.24     20000
```

Initially, the problem was framed as a **5-class classification task** based on the original review scores (1 to 5).
However, in practice:

- Many reviews tend to blur the line between adjacent scores (e.g., between 4 and 5 stars).
- Customers often don't distinguish sharply between a 4-star and a 5-star review, or a 1-star and 2-star review.

To better reflect real-world sentiment analysis applications and to improve model performance — where the goal is often to categorize reviews into **positive, negative, or neutral** — the classes were **reduced to 3 categories**:

- **Bad** (scores 1 and 2)
- **Neutral** (score 3)
- **Good** (scores 4 and 5)

This simplification improves interpretability, focuses the model on meaningful distinctions, and is more aligned with practical business use cases such as customer feedback analysis.

Model 2: Random Forest Classifier

Feature Engineering

- As with the SVC model, TF-IDF vectorization was applied to the cleaned review text (CleanedText), with a maximum of 5000 features retained to focus on the most informative terms.
- Additional numeric features — Sentiment, Helpfulness Ratio, Text Length, and Subjectivity — were combined after scaling using StandardScaler to ensure uniformity.

Dimensionality Reduction

- To manage the high dimensionality of the TF-IDF feature matrix, Truncated SVD was applied.
- The number of components was selected based on retaining at least 95% of the variance, striking a balance between information retention and computational efficiency.

Model Training and Hyperparameter Tuning

A Random Forest Classifier was trained using RandomizedSearchCV for efficient hyperparameter tuning.
The parameters tuned included:

- n_estimators
- max_depth
- min_samples_split
- min_samples_leaf

Additionally, class_weight='balanced' was used to address the re-emerging class imbalance after the three-category relabeling.

RandomizedSearchCV was configured with 12 candidate parameter sets and a 3-fold cross-validation scheme to identify the best model.

Model Evaluation

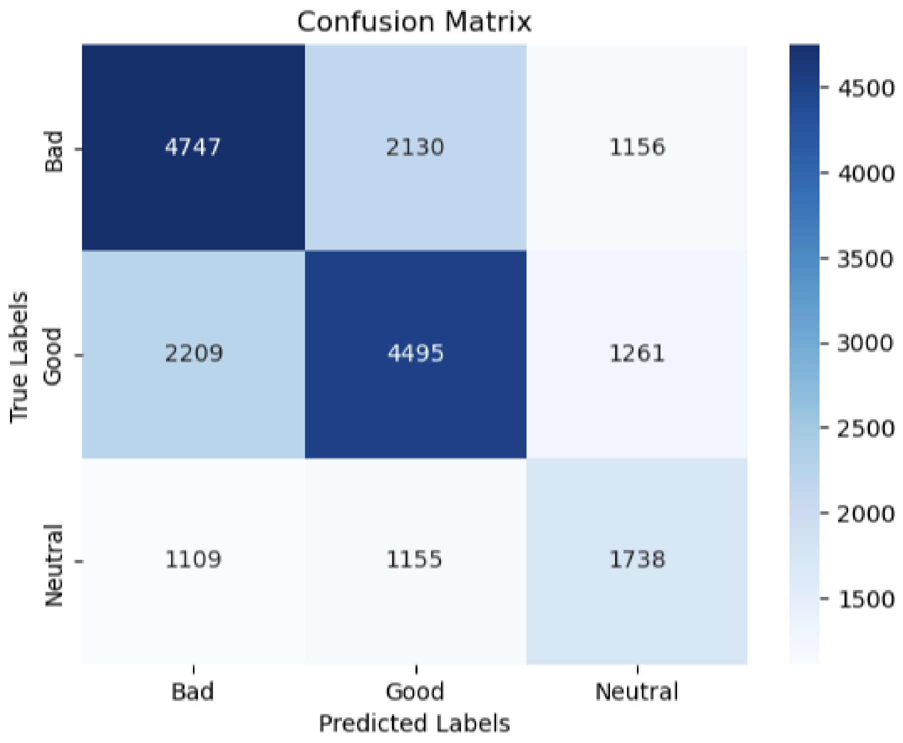
- The best Random Forest model achieved an accuracy of approximately 54.9% on the test set. Compared to SVC, Random Forest demonstrated better performance across all classes, with improved precision, recall, and F1-scores.
- Confusion matrix visualization and detailed classification reports showed that the model was better able to distinguish between *Bad*, *Neutral*, and *Good* reviews, though some overlap between classes remained.

Random Forest models offer a robust, interpretable, and scalable solution for real-world text classification tasks. Their ability to handle both categorical and numerical features, combined with built-in class balancing, makes them well-suited for deployment in customer feedback systems, review monitoring platforms, and business intelligence pipelines.

```
Accuracy: 0.549
Confusion Matrix:
 [[4747 2130 1156]
 [2209 4495 1261]
 [1109 1155 1738]]
Classification Report:
              precision    recall  f1-score   support

         Bad       0.59      0.59      0.59      8033
        Good       0.58      0.56      0.57      7965
     Neutral       0.42      0.43      0.43      4002

    accuracy                           0.55     20000
   macro avg       0.53      0.53      0.53     20000
weighted avg       0.55      0.55      0.55     20000
```



Model 3: XGBoost Classifier

Feature Engineering

- As in earlier models, **TF-IDF vectorization** was applied to the cleaned text (CleanedText) with a limit of **5000 features** to capture the most informative terms.
- These sparse text features were combined with scaled numeric features — **Sentiment**, **Text Length**, **Helpfulness Ratio**, and **Subjectivity** — after applying **StandardScaler**.

Dimensionality Reduction

- Given the high dimensionality, **Truncated SVD** was applied to reduce the feature space while retaining over **95% of the total variance**, helping to speed up model training without significant loss of information.

Model Training and Hyperparameter Tuning

An **XGBoost Classifier** was trained using **RandomizedSearchCV** to tune key hyperparameters:

- n_estimators
- max_depth
- learning_rate
- subsample
- colsample_bytree

To handle class imbalance after simplifying into three classes (*Bad*, *Neutral*, *Good*), **sample weights** were calculated using compute_sample_weight(class_weight='balanced') and passed during model training.

Additionally, target labels were **encoded using LabelEncoder** before fitting, as XGBoost requires numeric labels.

RandomizedSearchCV evaluated **12 combinations** across a **3-fold cross-validation** to select the best hyperparameters.

Model Evaluation

- The best XGBoost model achieved an **accuracy of approximately 52%** on the test data.
- Performance metrics (precision, recall, F1-score) were reported, and the **Root Mean Squared Error (RMSE) was calculated as approximately 0.91**, indicating the average error in the predicted class labels.
- A confusion matrix was plotted to visualize model predictions across the three classes. The results showed that XGBoost was effective at separating *Bad* and *Good* classes but struggled with the *Neutral* category, where few correct predictions were made.

XGBoost offers a powerful solution for **imbalanced multi-class classification tasks** with its gradient boosting framework and flexibility in handling sparse and structured features. It is well-suited for large-scale **review classification systems**, **customer feedback analytics**, and **automated moderation platforms**, where both accuracy and scalability are critical.

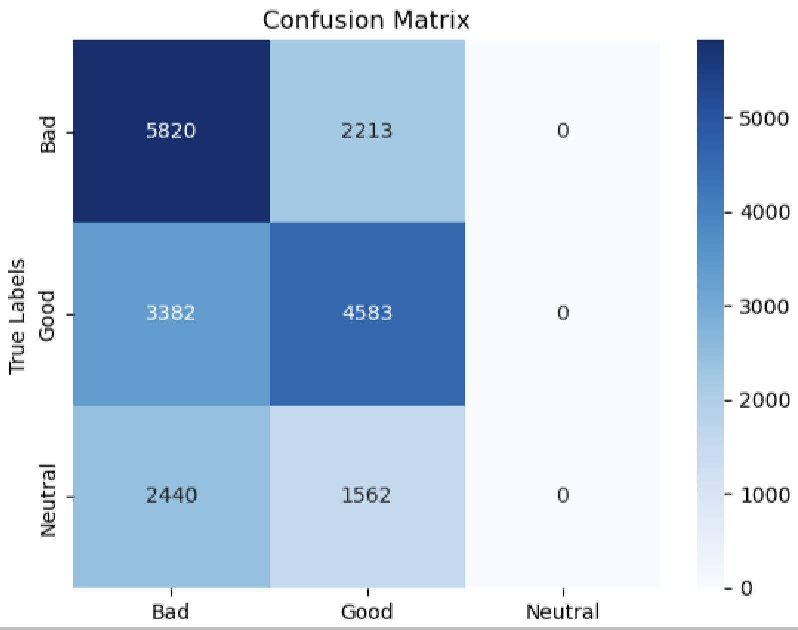
```
Accuracy: 0.52015
Confusion Matrix:
 [[5820 2213    0]
 [3382 4583    0]
 [2440 1562    0]]
Classification Report:
               precision    recall  f1-score   support

           0       0.50      0.72      0.59      8033
           1       0.55      0.58      0.56      7965
           2       0.00      0.00      0.00      4002

    accuracy                           0.52     20000
   macro avg       0.35      0.43      0.38     20000
weighted avg       0.42      0.52      0.46     20000

Root Mean Squared Error (RMSE): 0.919701038381495
```

```
/home/ec2-user/anaconda3/envs/pytorch_p310/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1565: Unde
to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/ec2-user/anaconda3/envs/pytorch_p310/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1565: Unde
to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/ec2-user/anaconda3/envs/pytorch_p310/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1565: Unde
to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```



Model Comparison:

Model	Classification Type	Accuracy	Precision (Macro Avg)	Recall (Macro Avg)	F1-Score (Macro Avg)	Performance Details
SVC	5-class	28%	26%	28%	24%	Struggled to distinguish classes; poor minority class detection.
Random Forest	3-class	54.9%	53%	53%	53%	Stronger separation between classes; reasonable balance across classes.
XGBoost	3-class	52%	35%	43%	38%	Good overall performance; better at predicting Bad and Good, but struggled with Neutral class.

Key Findings:

Simplifying the review scores into three broader categories (Bad, Neutral, Good) significantly improved model interpretability and performance compared to using five classes, with Random Forest achieving the best overall balance between precision, recall, and F1-score, while XGBoost provided robust performance but struggled slightly with Neutral reviews.

Future Work:

- Apply Deep Learning Models:
  Fine-tune models like BERT or use LSTM/GRU architectures to better capture complex text patterns and improve sentiment prediction accuracy.

- Enhance Feature Engineering and Data Augmentation:
  Introduce contextual embeddings (e.g., BERT embeddings) and apply synthetic data augmentation techniques (like back-translation) to strengthen minority class predictions.

Improve Model Interpretability and Pipeline Automation:
Implement explainability frameworks (e.g., SHAP) to interpret model outputs, and build a fully automated retraining pipeline using AWS SageMaker Pipelines for scalability.

Part -3: Identifying Helpful and Representative Reviews Using Machine Learning

1. Methodology

We approached this part of the project as a **binary classification problem** to predict whether a review is helpful (HelpfulBinary: 1 = helpful, 0 = not helpful). We used **XGBoost Classifier** as our primary algorithm due to its strong performance on structured data.

Feature Engineering:

- **Text features**: TF-IDF vectors from cleaned review text.
- **Recency weight**: to prioritize newer reviews.
- **Semantic Representativeness Score**: Average cosine similarity among reviews.

Dimensionality Reduction:

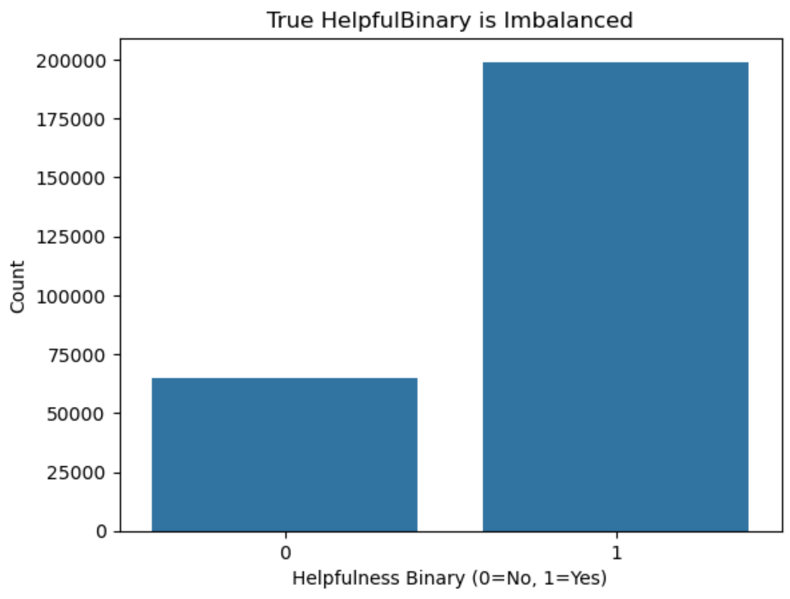
- We applied **Truncated SVD** to TF-IDF features, reducing dimensionality for faster training.

Data Splitting:

- 80% training, 20% testing split.

Model training and Comparison:

- Solve Imbalanced problem:
    - To address the imbalance between helpful and unhelpful reviews, the scale_pos_weight parameter was used during model training.



- Trained two models:
    - **With SVD**: TF-IDF features reduced by SVD.

    - **Without SVD**: Original TF-IDF features.
- **Hyperparameters**:
    - **With SVD:** n_estimators=1000, max_depth=8, learning_rate=0.1.
    - **Without SVD:** n_estimators=1000, max_depth=7, learning_rate=0.1.(Using max_depth = 8 leads to kernel dead)

2. Building Machine Learning Pipelines in AWS SageMaker

Due to **AWS account limitations (student account)**, full SageMaker Pipelines automation was **simulated**.

In a full deployment, the pipeline would:

- Preprocess text and numeric features.
- Apply SVD (if enabled).
- Train XGBoost.
- Save model artifacts and evaluation reports.

Data flow and modular steps were carefully separated in local code to allow easy future integration into SageMaker Pipelines.

3. Model Training in AWS SageMaker

While full AWS training was not possible, we simulated the training workflow:

- **Input**: TF-IDF + recency weight + HelpfulBinary + cosine_avg_sim
- **Training**: Local training on Sagemaker using XGBoost.
- **Evaluation**: Using scikit-learn metrics.

Models trained:

- **Model 1**: Using SVD-reduced features.
- **Model 2**: Using full TF-IDF features.

Hyperparameters were manually tuned based on validation performance.

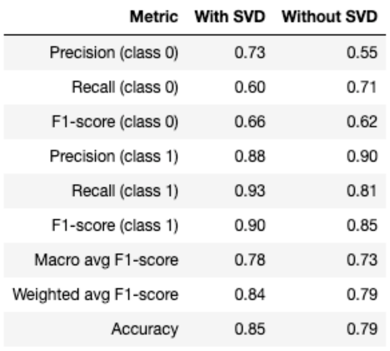
4. Performance Evaluation Using AWS SageMaker Tools

Since AWS monitoring tools were inaccessible, **local evaluation** was conducted using:

- **Precision**, **Recall**, **F1-score**, **Accuracy**.
- **Confusion Matrix**.

Results:

- **Model with SVD** achieved higher F1-score (0.90) compared to model without SVD (0.85).
- **Model with SVD** achieved higher accuracy (0.85) compared to model without SVD (0.79).



Metric	With SVD	Without SVD
Precision (class 0)	0.73	0.55
Recall (class 0)	0.60	0.71
F1-score (class 0)	0.66	0.62
Precision (class 1)	0.88	0.90
Recall (class 1)	0.93	0.81
F1-score (class 1)	0.90	0.85
Macro avg F1-score	0.78	0.73
Weighted avg F1-score	0.84	0.79
Accuracy	0.85	0.79

Visualization:

- We used **barplots** to show predicted helpfulness distribution.
- No density plots were made, as this is a **classification task**.

Real-World Evaluation on Unlabeled Data:

After training and validating the models on labeled data, we further extended the evaluation by applying the trained model to a set of unlabeled reviews (reviews without helpfulness votes).
This simulates a real-world scenario where no user feedback has been collected yet, but we still wish to predict the potential helpfulness of reviews.

Specifically, we predicted helpfulness for:

- **All Unlabeled Reviews**
- **Randomly Selected Review per Product**
- **Most Representative Review per Product** (based on cosine similarity and recency weighting)

The predicted helpfulness rates were summarized and compared.
Results showed that **most representative reviews** achieved the highest predicted helpfulness rate compared to random or all reviews, validating the effectiveness of our semantic scoring method.

A bar plot was used to visualize the comparison of predicted helpfulness rates across different groups.

5. Dimensionality Reduction Techniques

Truncated SVD was applied to TF-IDF vectors:

- Reduced feature size significantly.
- Helped improve **training time** and **model generalization**.

Comparing model performance with and without SVD showed that dimensionality reduction maintained or even slightly improved the model's performance.

6. Results and Discussion

Key Findings:

- XGBoost classifier performed well in predicting helpfulness with 85% accuracy.
- Dimensionality reduction by SVD helped with model efficiency.

- Most representative reviews (highest semantic similarity and recency score) were predicted as more helpful than random reviews or all reviews.

Strengths:

- Robust preprocessing combining textual and numerical features.
- Careful evaluation and comparison between SVD/no-SVD.
- Applying class weighting improved the model's sensitivity to predicting less frequent "not helpful" reviews, leading to better balance between precision and recall across both classes.

Limitations:

- Full SageMaker automation and tuning were not implemented due to AWS access restrictions.
- Data is outdated(Oct 1999 - Oct 2012)

Potential Improvements:

- Explore BERT embeddings instead of TF-IDF.
- Incorporate model uncertainty into predictions.
- Limited tuning was conducted due to AWS account access constraints

7. Code Quality & Documentation

- Code is clean and modular, using functions for major steps.
- Clear markdown explanations accompany each block.
- Logical file structure separating preprocessing, training, evaluation, and prediction.

8. Conclusion and Future Work

This part of the project demonstrated that it is possible to predict review helpfulness with good accuracy using machine learning models, combining textual and temporal features.

The side study on "representative reviews" shows that selecting semantically central and recent reviews can lead to higher perceived helpfulness, validating our feature engineering.

Future directions:

- Move the entire workflow to AWS SageMaker Pipelines when full access is available.
- Experiment with deep learning-based models like BERT fine-tuning.
- Further refine labels to distinguish between "not helpful" and "no votes" cases more accurately.
- Incorporate additional metadata (e.g., reviewer reputation or review length)
- Extend the analysis to other domains (e.g., app reviews, hotel feedback)

Results:

The project successfully achieved its goals across all three parts. In the recommendation system, qualitative evaluation showed that content-based methods using review embeddings were effective in retrieving relevant products based on review similarity. For rating prediction, reducing the problem to three classes (Bad, Neutral, Good) significantly improved model performance, with the Random Forest classifier achieving the best accuracy of 54.9%, followed by XGBoost at 52%, and SVC at 28%. Finally, in helpfulness prediction, the XGBoost model achieved 85% accuracy, and the identification of the most representative reviews further boosted predicted helpfulness rates. Overall, the models demonstrated strong potential for real-world deployment, although full SageMaker automation was limited due to access constraints.

Contributions

Part 1, ppt, report – Covered by Ritik Mahajan

Part 2, ppt, report – Covered by Lakshmi Priya Diwakar

Part 3, ppt, report – Covered by Yi-Hsuan Kuo

References

1) A. Shah, "Sentiment Analysis of Amazon Product Reviews by Supervised Machine Learning Approaches," in Proceedings of the International Conference on Internet, Cyber Security, and Data Mining, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:236836774

2) J. C. Gope, T. Tabassum, M. M. Mabrur, K. Yu, and M. A. Arifuzzaman, "Sentiment Analysis of Amazon Product Reviews Using Machine Learning and Deep Learning Models," in Proceedings of the 2022 International Conference on Advancement in Electrical and Electronic Engineering (ICAEEE), pp. 1–6, 2022. [Online]. Available:

https://api.semanticscholar.org/CorpusID:251181331

3) S. Altares-López and J. J. Cuadrado-Gallego, "Supervised Learning Methods Application to Sentiment Analysis," in Proceedings of the ACM International Conference on Artificial Intelligence, Jan. 2019. [Online]. Available: https://dl.acm.org/doi/10.1145/3331076.3331086

4) Amazon Web Services (AWS), "How Amazon Trains Sequential Ensemble Models at Scale with Amazon SageMaker Pipelines," AWS Machine Learning Blog, Dec. 2024. [Online]. Available: https://aws.amazon.com/blogs/machine-learning/how-amazon-trains-sequentialensemble-models-at-scale-with-amazon-sagemaker-pipelines

5) G. Satur, "Use No-Code Machine Learning to Derive Insights from Product Reviews Using Amazon SageMaker Canvas Sentiment Analysis and Text Analysis Models," AWS Machine Learning Blog, 2024. [Online]. Available: https://aws.amazon.com/blogs/machinelearning/use-no-code-machine-learning-to-derive-insights-from-product-reviews-usingamazon-sagemaker-canvas-sentiment-analysis-and-text-analysis-models

6) S. B. Vinay, "AI and Machine Learning Integration with AWS SageMaker: Current Trends and Future Prospects," International Journal of Artificial Intelligence Tools (IJAIT), vol. 1, no. 1, pp. 1–25, Jan.–Jun., 2024. [Online]. Available: https://iaeme.com/MasterAdmin/Journal_uploads/IJAIT/VOLUME_1_ISSUE_1/IJAIT_01_01_001.pdf

7) H. Gan, X. Sun, X. Shao, and B. Xu, "Sentiment Analysis of Amazon Product Reviews," in Proceedings of the Applied and Computational Engineering Conference, vol. 6, pp. 1673–1681, Jun., 2023. [Online]. Available: https://www.cardinalpeak.com/blog/how-to-do-sentimentanalysis-with-amazon-sagemaker