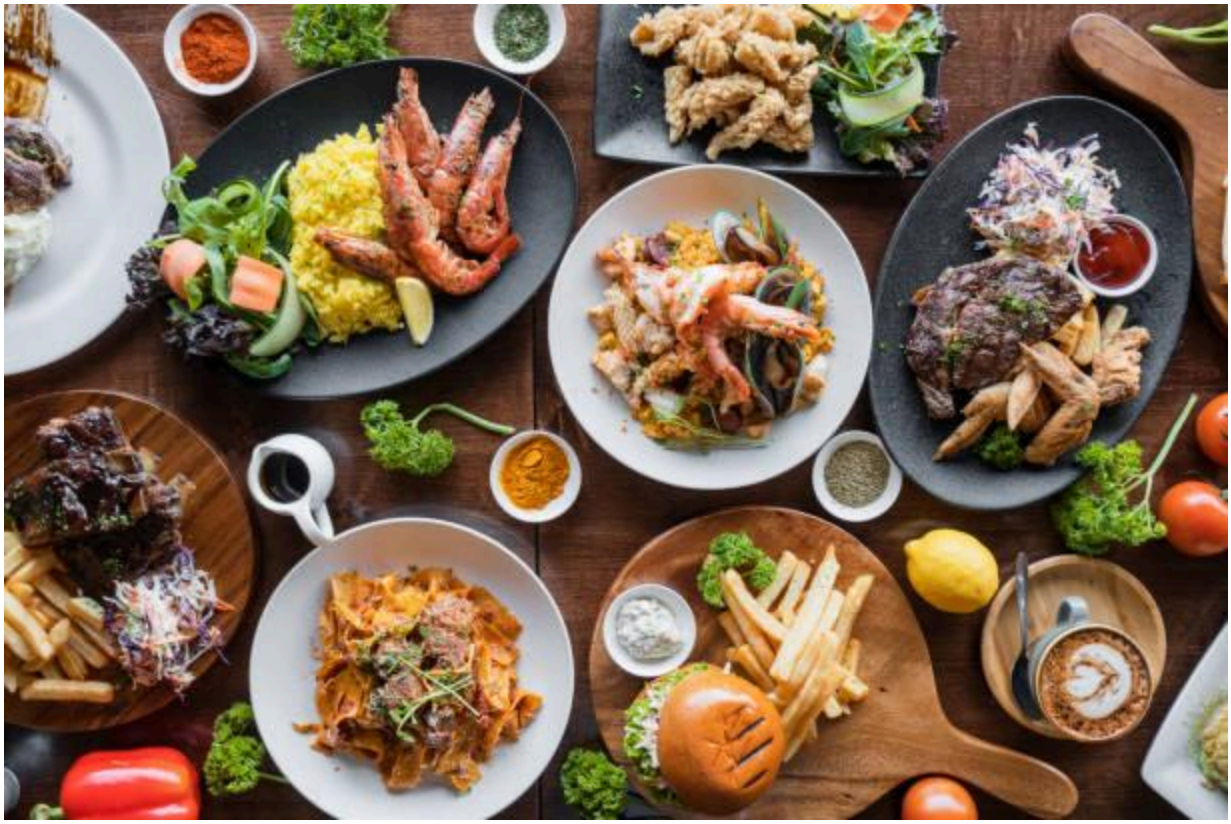
ML Techniques I

Sentiment Analysis Report

Amazon Fine Food Reviews



Group Members

- Nina Mwangi
- Aubin Ntwali
- Pascal Mugisha
- Ruth Iradukunda

Dataset Link: https://www.kaggle.com/datasets/snap/amazon-fine-food-reviews

Github link: https://github.com/NinaMwangi/Sentiment_Analysis.git

Introduction.

This project is based on fine food reviews from Amazon. The data spans a period of more than 10 years, including all ~500,000 reviews up to October 2012. Reviews include product and user information, ratings, and a plain text review. It also includes reviews from all other Amazon categories. Our main focus, however, was on the score and text reviews columns.

Exploratory Data Analysis.

To begin our EDA, we checked the shape of the data and downsized to 20,000 rows. We then checked for missing values and established that the score and text columns had no missing values. We also checked for the number of times each score occurred.

Data Visualization

Bar Plot: Our first visualization is a bar plot displaying the count of reviews by score. This plot revealed that there are more 5-star reviews in this dataset compared to the other stars. This dataset has a positive bias; we also observed an uptick in the 4-star and 1-star reviews, and noted that the lowest count is in the 2-star rating [1].

Word Cloud: We preprocessed the text column in preparation for the word cloud visualisation. First, we joined the text column into one big string, then we removed all punctuation and numbers, converted all text to lowercase, removed stop words, and lemmatized the text [2].

Preprocessing and Feature Engineering:

Label Encoding Sentiment.

We performed semantic mapping from the score column to three sentiment categories: “positive”, “negative”, and “neutral”. We engineered this column so that we can have a multiclass sentiment classification. We then used the `LabelEncoder()` function to label encode the sentiment strings to integers and stored them in the `label` column. After which, we one-hot encoded the label column in preparation for training the model [3],[4].

Preprocessing Individual Reviews.

The initial preprocessing was done on the big string that had been created to help in creating the word cloud. However, now we are preprocessing each review and storing it in a new column titled `clean_text`, using the preprocess function that removes punctuation, numbers, splits the string, removes stop words, and lemmatizes the text [2].

Tokenizing and Padding

In our next step, we tokenized the `clean_text` column using `Tokenizer()`, and we then created a sequence and padded the sequence to ensure that the inputs are all of the same size. We then split the data into train and test sets [3],[4].

Word Embedding using GloVe

We used GloVe for our Embeddings, particularly `glove.6B.100d.txt,` for a good balance in size and performance. We then created the Embedding matrix and proceeded to build our models.

MODELS

Traditional ML Models

Linear Regression

Text classification model, we tried to use Logistic regression due to its effectiveness with high-dimensional sparse data like TF-IDF, which works well for binary and multiclass text classification. We used TF-IDF to transform the cleaned text into numerical vectors, capturing the importance of words while reducing the influence of commonly used words. The resulting features are split into training and testing sets to evaluate the performance of the model. After carefully evaluating the model with an accuracy of **84.2%,** we tried to improve the model performance, hyperparameter tuning by applying `GridSearchCV`, where we explored a combination of regularization type `L2`, solvers, and penalty strengths to find the optimal configuration. After fine-tuning our model, the accuracy improved up to **85.72%**.

Naive Bayes

We implemented a **Multinomial Naïve Bayes** model using TF-IDF vectorized features extracted from the reviews dataset. Initially, the model showed strong performance on positive sentiment classification but struggled to identify negative and neutral sentiments due to class imbalance and limited feature richness.

To address this, we incrementally applied several improvements:

1. TF-IDF Optimization:
    - Used `ngram_range (1, 2)` to capture bi-grams like “not good”.
    - Applied `min_df =3` and `max_df = 0.9` to remove rare and overly frequent words.
    - Limited to the top `5000` features for efficiency.
2. Model Smoothing:
    - Applied smoothing with `alpha=0.5` to prevent zero probability errors for rare terms and improve generalization.

Final Results

Sentiment	Precision	Recall	F1-Score	Support
Negative	0.84	0.31	0.45	599
Neutral	0.48	0.09	0.15	342
Positive	0.81	0.99	0.89	3059
Accuracy	-	-	0.808	4000
Micro Avg.	0.71	0.46	0.50	-
Weighted Avg	0.79	0.81	0.76	-

The model performs very well on positive reviews, achieving high precision (0.81) and near-perfect recall (0.99). Performance on negative reviews improved significantly compared to earlier iterations, achieving a 31% recall, up from 17%. Neutral reviews remain a challenge, with low recall (0.09). The overall macro F1-score of 0.50 indicates moderate balance across all classes, while the weighted F1-score of 0.76 reflects strong performance skewed toward the dominant class (Positive).

SVM

We implemented a Support Vector Machine (SVM); the model was trained using the same TF-IDF features used in the Naïve Bayes model to ensure a fair comparison.

To handle class imbalance, we configured the SVM with:

- `class_weight='balanced'`: to penalize misclassification of underrepresented classes more.
- `C=0.8`: a slightly relaxed regularization parameter to reduce overfitting and improve generalization.

We adjusted the SVM model further by applying `class_weight='balanced'` and retraining the model:

Sentiment	**Precision**	**Recall**	**F1-Score**	**Support**
Negative	0.63	0.64	0.64	599
Neutral	0.33	0.33	0.33	342
Positive	0.91	0.91	0.91	3059
Accuracy	-		0.817	4000
Macro Avg	0.62	0.63	0.62	-
Weighted Avg	0.82	0.82	0.82	-

After class balancing, the model showed more uniform recall across all classes, particularly improving Neutral detection, with all F1-scores exceeding 0.30. Although there was a slight drop in overall accuracy (from 83.4% to 81.7%), the **class-wise fairness** improved, an important consideration for real-world use.

Deep Learning Models

LSTM

We developed a deep learning model using an LSTM-based neural network to classify text data into three sentiment categories: Negative, Neutral, and Positive. Text is tokenized, padded, and split into training and test sets with stratified labels for maintaining balanced classes. The model consists of an embedding layer, two stacked LSTM layers (**64 units** and **32 units**), dropout layers to prevent overfitting, and dense layers ending with a `softmax` activation for multi-class classification. It is compiled using the Adam optimizer with a custom learning rate and trained under early stopping in order to prevent overfitting with a 20% validation split. Model performance is evaluated using a confusion matrix, classification report, and test accuracy. The training and validation loss curves are also plotted to observe learning behavior for epochs.

GRU

To evaluate model performance, we used the following metrics:

- **Accuracy**: Overall proportion of correct predictions.
- **F1-Score**: Harmonic mean of precision and recall—especially important for imbalanced classes.
- **Confusion Matrix**: Visual representation of prediction errors across all three sentiment classes.

Classification Report:

	precision	recall	f1-score	support
negative	0.63	0.62	0.62	589
neutral	0.00	0.00	0.00	330
positive	0.86	0.96	0.91	3081

```
    accuracy                           0.83      4000
   macro avg       0.50      0.53      0.51      4000
weighted avg       0.76      0.83      0.79      4000
```

The model performs exceptionally well on positive sentiment but struggles with the neutral class, likely due to class imbalance or label noise.

Configuration	**Learning Rate**	**Early Stopping**	**Accuracy**
GRU (128 units), Dropout=0.5	default	No	76.15%
GRU (128 units), Dropout=0.5	default	yes(patience 3)	77.07%
Bidirectional Gru(128)	default	yes(patience 3)	82.53%
Stacked BiGRU(128, 64)	default	yes(patience 3)	82.38%
Stacked BiGRU(128, 64)	0.0005	yes(patience 3)	82.88%
BiGRU(128)	0.0001	yes(patience 3)	83.22%

Conclusion

The data underperformed on the neutral class, which was an indication of data imbalance; however, from the beginning, we had noticed that the data was biased towards the positive reviews. During training, despite tuning, adding more GRU layers did not improve accuracy, suggesting possible diminishing returns.

Overfitting was mitigated by:

- Adding Dropout.
- Using EarlyStopping (patience = 3).
- Reducing the learning rate.

Reference:

[1] K. Hu, M. A. Bakker, S. Li, and T. Kraska, "VizML: A Machine Learning Approach to Visualization Recommendation," in Proc. ACM Human Factors in Computing Systems Conf. (CHI), 2019. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3290605.3300358

[2] P. B. Bafna and J. R. Saini, "Marathi Text Analysis Using Unsupervised Learning and Word Cloud," 2020. [Online]. Available: https://www.academia.edu/download/89735173/C4727029320.pdf

[3] S. K. Das and M. Z. Rahman, "A Study on Machine Learning Algorithms with Different Encoding Techniques for Identifying the Right One for Patients' Big Data," Journal of Science, Jahangirnagar University, vol. 44, no. 1, pp. 47–56, 2021. [Online]. Available: https://jos.ju-journal.org/jujs/article/view/59

[4] A. Erkan and T. Güngör, "Analysis of Deep Learning Model Combinations and Tokenization Approaches in Sentiment Classification," IEEE Access, vol. 11, pp. 125353–125368, 2023. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10332170/