NLP 220 Assignment 1

Soren Larsen
UCSC Silicon Valley Extension
Santa Clara, CA 95054
snlarsen@ucsc.edu

Abstract

This report documents NLP 220 Assignment 1, which involves implementing and evaluating machine learning models for text classification using the Amazon Book Reviews dataset. The assignment is divided into three parts. Part A focuses on feature engineering and applying three distinct features—Bag-of-Words, TF-IDF, and One-Hot Encoding—across Naive Bayes, SVM, and Decision Tree classifiers. These models are compared based on accuracy, F1 scores, and confusion matrices, with additional analysis of training and inference times.

Part B extends the sentiment analysis task to Stanford's movie review corpus, requiring n-gram features and the evaluation of five different models for sentiment classification. The models are fine-tuned on a validation set and tested on a held-out test set.

Part C requires implementing a custom Naive Bayes classifier and comparing its performance against the scikit-learn implementation on the Amazon dataset. This report provides insights on the models' performance, feature contributions, and recommendations based on accuracy and efficiency.

1 Dataset Format and Tasks

The dataset used for this assignment contains reviews of books from Amazon. It is structured with the following key columns:

- **review/text:** Contains the content of the review.
- **review/score:** Holds the score given to the book (on a scale of 1-5 stars).
- **Title:** Refers to the title of the book being reviewed.
- **review/summary:** Contains a brief summary or title of the review.

Other columns in the dataset may also provide useful information for learning to predict scores. While the primary focus is on processing the `review/text` column, additional features may be derived from other columns or combinations (e.g., using the concatenation of `review/summary` and `review/text`).

1.1 Dataset Tasks and Deliverables

The primary objective is to create a binary classification dataset following these rules:

- **Positive Reviews:** Any review with a score of 4 or higher is considered positive.
- **Negative Reviews:** Any review with a score of 2 or lower is considered negative.
- **Ignore Neutral Reviews:** Reviews with a score of 3 are excluded from the dataset.

We performed a `train/test` split using the `train_test_split` method from `sklearn.model_selection`. The data was shuffled using a fixed random seed to ensure reproducibility, with 85% allocated to the training set and 15% to the test set.

The class distributions for the training and test datasets are included in Figure 1 in the appendix.

2 Feature Engineering for Naive Bayes, SVM & Decision Tree

2.1 Naive Bayes

Naive Bayes is a simple yet powerful probabilistic classifier based on Bayes' theorem, which assumes that features are independent of each other, making it particularly effective for text classification tasks.

2.1.1 Feature Descriptions

- **Bag-of-Words (BoW) + Multinomial Naive Bayes:** BoW converts text into word frequency counts, and Multinomial Naive Bayes

models these counts assuming each word independently contributes to the class. This approach works well when frequent words are strong indicators of class labels (e.g., "great" for positive).

- **TF-IDF + Multinomial Naive Bayes:** TF-IDF assigns weights to words based on their importance across the dataset, emphasizing rare but meaningful words. Multinomial Naive Bayes integrates these features effectively by leveraging the weighted word counts to improve classification accuracy.

- **One-Hot Encoding + Multinomial Naive Bayes:** One-Hot Encoding converts categorical data (like titles and review summaries) into binary vectors, allowing each category to be represented independently. This encoding works well for categorical features with a limited number of unique values, providing a clear structure for the Naive Bayes classifier to learn from.

2.1.2 Naive Bayes Evaluations

Model	Accuracy	Macro-F1	Binary F1	Training Time (s)	Prediction Time (s)
BoW	0.8502	0.8497	0.8410	0.0102	0.0028
TF-IDF	0.8709	0.8708	0.8670	0.0063	0.0018
One-Hot Encoding	0.8578	0.8577	0.8598	0.0024	0.0002

Table 1: Accuracy, Macro-F1, Binary F1 Scores, and Training/Prediction Times for Naive Bayes Models with Different Features

BoW Model	Predicted Negative	Predicted Positive
Actual Negative	2044	206
Actual Positive	468	1782

Table 2: Confusion Matrix for BoW Model

TF-IDF Model	Predicted Negative	Predicted Positive
Actual Negative	2026	224
Actual Positive	357	1893

Table 3: Confusion Matrix for TF-IDF Model

One-Hot Encoding Model	Predicted Negative	Predicted Positive
Actual Negative	1898	352
Actual Positive	288	1962

Table 4: Confusion Matrix for One-Hot Encoding Model

2.2 SVM

Support Vector Machines (SVMs) are powerful classifiers that aim to find the optimal hyperplane to separate data points from different classes. SVMs are particularly effective for high-dimensional datasets and can be enhanced using different kernels.

2.2.1 Feature Descriptions

- **Bag-of-Words (BoW) + SVM:** BoW converts text into word frequency counts, and the SVM model separates the classes using these counts. It works well when frequent words serve as strong class indicators (e.g., "excellent" for positive sentiment).

- **TF-IDF + SVM:** TF-IDF assigns weights to words based on their importance across the dataset, and the SVM model uses these weighted vectors to separate classes effectively.

- **One-Hot Encoding + SVM:** One-Hot Encoding transforms categorical data into binary vectors, enabling the SVM to effectively distinguish classes based on unique categorical information such as titles or product summaries.

2.2.2 SVM Evaluations

Model	Accuracy	Macro-F1	Binary F1	Training (s)	Prediction (s)
BoW	0.8587	0.8587	0.8588	853.1412	12.8247
TF-IDF	0.8902	0.8902	0.8898	254.0640	20.9949
One-Hot	0.8558	0.8558	0.8575	23.9038	1.1523

Table 5: Accuracy, Macro-F1, Binary F1 Scores, and Training/Prediction Times for SVM Models with Different Features

BoW Model	Predicted Negative	Predicted Positive
Actual Negative	1930	320
Actual Positive	316	1934

Table 6: Confusion Matrix for BoW + SVM Model

TF-IDF Model	Predicted Negative	Predicted Positive
Actual Negative	2011	239
Actual Positive	255	1995

Table 7: Confusion Matrix for TF-IDF + SVM Model

One-Hot Encoding Model	Predicted Negative	Predicted Positive
Actual Negative	1899	351
Actual Positive	298	1952

Table 8: Confusion Matrix for One-Hot Encoding + SVM Model

2.3 Decision Tree

Decision Trees are interpretable and efficient models that split data based on feature values to create a hierarchy of decisions. They handle both categorical and numerical data well and are suitable for feature-rich datasets.

2.3.1 Feature Descriptions

- **Bag-of-Words (BoW) + Decision Tree:** BoW converts text into word frequency counts, and the Decision Tree model uses these counts to perform splits. This feature helps the model identify frequently occurring terms that serve as indicators of class labels.
- **TF-IDF + Decision Tree:** TF-IDF assigns weights to words based on their importance across the dataset, allowing the Decision Tree to prioritize important terms during splitting.
- **One-Hot Encoding + Decision Tree:** One-Hot Encoding transforms categorical data (such as product titles or summaries) into binary vectors. The Decision Tree uses these binary vectors to perform splits, effectively distinguishing between unique categorical attributes.

2.3.2 Decision Tree Evaluations

Model	Accuracy	Macro-F1	Binary F1	Training (s)	Prediction (s)
BoW	0.7084	0.7084	0.7071	13.3904	0.0027
TF-IDF	0.7051	0.7051	0.7048	17.3561	0.0021
One-Hot	0.8480	0.8480	0.8485	3.1791	0.0065

Table 9: Accuracy, Macro-F1, Binary F1 Scores, and Training/Prediction Times for Decision Tree Models with Different Features

BoW Model	Predicted Negative	Predicted Positive
Actual Negative	1604	646
Actual Positive	666	1584

Table 10: Confusion Matrix for BoW + Decision Tree Model

TF-IDF Model	Predicted Negative	Predicted Positive
Actual Negative	1589	661
Actual Positive	666	1584

Table 11: Confusion Matrix for TF-IDF + Decision Tree Model

One-Hot Encoding Model	Predicted Negative	Predicted Positive
Actual Negative	1900	350
Actual Positive	334	1916

Table 12: Confusion Matrix for One-Hot Encoding + Decision Tree Model

3 Analysis

This section provides a comparative analysis of the nine models evaluated in this assignment. Each model was trained using three distinct feature engineering techniques—Bag-of-Words (BoW), TF-IDF, and One-Hot Encoding—applied to Naive Bayes, SVM, and Decision Tree classifiers. The results, presented in the earlier sections, are referenced here to discuss accuracy, macro-F1 scores, and confusion matrices, followed by an analysis of the training and prediction times.

3.1 Accuracy and F1 Score Comparison

The highest-performing model in terms of accuracy and F1 scores was the **SVM with TF-IDF features**, achieving an accuracy of 89.02% and a binary F1 score of 0.8898. This outperforms all other models, as shown in the earlier tables. The **One-Hot Encoding Decision Tree model** also performed well, with 84.8% accuracy, indicating that categorical features can be effective when combined with decision tree classifiers.

Naive Bayes models demonstrated competitive performance, with the **TF-IDF variant** achieving 87.09% accuracy. This confirms the effectiveness of TF-IDF in improving classification by assigning importance to rare but meaningful words. However, the **Decision Tree models with BoW and TF-IDF** did not perform as well, with accuracy scores around 70%, highlighting potential limitations of these models for high-dimensional text features.

The confusion matrices for each model (in the earlier section) reveal that most models struggled with correctly predicting negative reviews, likely due to class imbalance in the data. The Naive Bayes and SVM models with TF-IDF performed slightly better in this regard, suggesting that weighted features help differentiate between positive and negative sentiment.

3.2 Training and Inference Time Comparison

As seen from the earlier tables, there is a significant variation in training and prediction times across the models. **Naive Bayes models** were the fastest to train and predict, with training times below 0.01 seconds and prediction times near 0.002 seconds. This makes them ideal for time-sensitive applications where rapid predictions are required.

In contrast, **SVM models** had considerably longer training times, with **BoW + SVM** taking over 850 seconds to train. While **TF-IDF + SVM** achieved the highest accuracy, it also required 254 seconds to train, with a prediction time of nearly 21 seconds. This indicates that while SVM models provide high accuracy, they may not be suitable for scenarios where rapid training or real-time inference is required.

The **Decision Tree models** showed moderate performance in terms of speed. The **One-Hot Encoding + Decision Tree** model had one of the shortest training times at just 3.18 seconds, making it a viable option when both interpretability and efficiency are needed. The **BoW and TF-IDF Decision Trees** also performed well in terms of inference time, with predictions completed in around 0.002 seconds, but their lower accuracy makes them less favorable.

3.3 Recommended Model

Based on this analysis, the **SVM with TF-IDF features** is recommended for tasks where accuracy is the primary objective, as it outperformed all other models. However, this model comes with the trade-off of longer training and prediction times, making it suitable for applications where computational resources and time are not constraints.

For tasks where **efficiency** is prioritized, **Naive Bayes with TF-IDF features** offers a reasonable balance between accuracy and speed. It achieves competitive performance with minimal training and prediction times, making it a good fit for real-time applications.

The **Decision Tree with One-Hot Encoding** is recommended when interpretability is important. This model offers competitive accuracy, fast training, and ease of understanding, making it suitable for applications where transparency is required, such as decision support systems.

3.4 Conclusion

In summary, the feature engineering techniques applied in this assignment highlight the trade-offs between accuracy and computational efficiency. TF-IDF consistently led to better performance across models, demonstrating its utility in text classification. While SVM models achieved the highest accuracy, Naive Bayes models provided the best balance of speed and performance. Decision Tree models with One-Hot Encoding also showed promise, particularly for interpretable applications. These insights can guide future model selection based on specific requirements such as accuracy, speed, or interpretability.

4 Sentiment Analysis on Stanford's Total Movie Review Corpus

The dataset used for Part B of this assignment is the **Large Movie Review Dataset**, compiled by Maas et al. (2011) (Maas et al., 2011). This dataset contains 50,000 movie reviews collected from IMDb, split evenly into two sets: **25,000 reviews for training** and **25,000 reviews for testing**. Each set is carefully balanced, containing an equal number of **positive** and **negative** reviews. Neutral reviews with ratings between 4 and 6 stars are excluded to ensure clear sentiment polarity (**?**).

4.1 Dataset Overview

The movie reviews in the dataset are labeled as follows:

- **Positive Reviews**: Ratings of 7 stars and above.
- **Negative Reviews**: Ratings of 4 stars and below.

Each review is provided as a separate text file, with the training and test data further divided into `pos/` and `neg/` folders to indicate sentiment labels.

4.2 Data Processing and Splitting

To facilitate model training, the reviews from the training data were **parsed and saved into structured CSV files**. The following preprocessing steps were applied:

1. **Parsing and Combining:** Reviews from the `pos/` and `neg/` folders in the `train/` directory were combined into a single DataFrame, with columns `id`, `review`, and `label`. Here, `id` is derived from the filename, and `label` takes

values 1 for positive reviews and 0 for negative reviews.

2. **Balanced and Stratified Splitting:** The combined training data was split into two subsets:
   - **Train Set (90%)**: Used to train the sentiment classification models.
   - **Validation Set (10%)**: Used to validate the model's performance during training.

   To maintain the balance between positive and negative reviews, a `stratified split` was performed, ensuring both the train and validation sets retained equal proportions of each class.

3. **Saving as CSV:** The processed splits were saved as `train.csv` and `validation.csv` for downstream experiments.

4.3 Test Data Preparation

The reviews from the `test/` directory were combined into a single structured DataFrame with the following columns:

- `id`: Extracted from the filename (e.g., `0_9.txt`).
- `review`: The full text of the review.
- `label`: The sentiment label, with 1 for positive and 0 for negative reviews.

This DataFrame was saved as `test.csv`. Both the review text and sentiment labels are included in this single file, allowing the model's predictions to be evaluated directly against the ground truth labels.

During evaluation, the trained model is applied to the reviews in `test.csv`. The predictions are saved in a new column called `predicted_label`. This augmented DataFrame is saved to a new file named `test_predictions.csv`, with the following structure:

- `review`: The original review text.
- `label`: The true sentiment label (1 for positive, 0 for negative).
- `predicted_label`: The predicted sentiment label.

This approach allows for straightforward comparison between the true and predicted labels, ensuring the model's performance can be easily assessed on unseen data.

5 Model Optimization and Evaluation

The implementation aimed to determine the optimal n-gram configuration for each of five machine learning models: Logistic Regression, SGD Classifier, Random Forest, Multinomial Naive Bayes, and Decision Tree. Each model was trained using a tailored n-gram configuration, reflecting the nuances of each algorithm. The goal was to leverage n-gram features effectively to achieve the highest validation accuracy while minimizing overfitting.

5.1 Optimal N-gram Configurations for Each Model

The n-gram configurations for each model were carefully selected to match the algorithm's strengths. Below is a summary of the configurations used:

- **Logistic Regression:**
  - N-gram Range: (1, 2)
  - Stop Words: English
  - Max Features: 10,000
  - min_df: 5
  - max_df: 0.75
  - **Reason:** Logistic Regression is a linear model that assumes feature independence, making it particularly effective when working with sparse feature matrices. Bigrams are crucial for capturing subtle contextual dependencies (e.g., "not good" vs. "good"), which improves sentiment detection. The removal of stop words ensures that the model focuses on more meaningful words rather than high-frequency, low-information terms such as "the" or "and." Limiting the vocabulary size (max features) to 10,000 prevents overfitting by discarding rare n-grams while retaining enough variance to improve model generalization.
- **SGD Classifier:**
  - N-gram Range: (1, 2)
  - Stop Words: English
  - Max Features: 20,000
  - min_df: 3
  - max_df: 0.8
  - **Reason:** The Stochastic Gradient Descent (SGD) Classifier is optimized for

large-scale and high-dimensional feature spaces. By using bigrams, the model captures more nuanced shifts in sentiment that single words cannot express (e.g., "very bad" vs. "bad"). Increasing the maximum features to 20,000 helps the model leverage more informative n-grams without underfitting. The smaller min_df ensures that rare but significant phrases are retained, while a higher max_df helps the model ignore n-grams that appear too frequently and provide little class discrimination (e.g., common filler phrases).

- **Random Forest:**
  - N-gram Range: (1, 1)
  - Stop Words: None
  - Max Features: 5,000
  - min_df: 2
  - max_df: 0.7
  - **Reason:** Random Forests work best when the feature space is simpler, as excessive dimensionality can lead to increased noise and overfitting. Therefore, using unigrams ensures that the model focuses on individual, high-impact words without adding complexity. Including stop words, which often play functional roles in sentence structure, can sometimes aid tree-based models by introducing patterns useful for decision splits. A smaller max feature size (5,000) and stricter filtering (min_df = 2) strike a balance between variance and generalization, helping the model avoid noise from rare words.
- **Multinomial Naive Bayes:**
  - N-gram Range: (1, 2)
  - Stop Words: English
  - Max Features: 5,000
  - min_df: 5
  - max_df: 0.9
  - **Reason:** Multinomial Naive Bayes is based on probabilistic assumptions, which makes it effective when features (e.g., words or phrases) provide meaningful class probabilities. Bigrams are useful because they allow the model to account for word sequences, which can alter the meaning significantly (e.g., "not bad" vs. "bad"). Limiting the features to 5,000 ensures the model is not overwhelmed by infrequent or redundant phrases, while setting max_df to 0.9 ensures that very frequent n-grams are not discarded if they carry useful class information. Stop words are removed to improve focus on more informative words.
- **Decision Tree:**
  - N-gram Range: (1, 1)
  - Stop Words: English
  - Max Features: 3,000
  - min_df: 10
  - max_df: 0.6
  - **Reason:** Decision Trees are prone to overfitting, especially with complex feature spaces. Therefore, restricting the model to unigrams ensures simplicity, and a small feature set (3,000) helps maintain interpretability while avoiding excessive branching. Removing common stop words reduces noise and makes the splits more meaningful. The higher min_df threshold filters out rare n-grams that could introduce noise, while a lower max_df ensures that very frequent words or phrases are discarded if they do not contribute to class differentiation.

5.2 Model Performance on the Validation Set

Each model was trained on the training set and evaluated on the validation split using its respective n-gram configuration. The details of the configurations and their corresponding validation accuracies are presented below:

- **Logistic Regression**
  - **N-gram Range:** (1, 2)
  - **Stop Words:** English
  - **Max Features:** 10,000
  - **Min DF:** 5, **Max DF:** 0.75
  - **Validation Accuracy:** 87.36%
- **SGD Classifier**
  - **N-gram Range:** (1, 2)
  - **Stop Words:** English
  - **Max Features:** 20,000

- **Min DF:** 3, **Max DF:** 0.8
- **Validation Accuracy:** 86.68%

- **Random Forest Classifier**
  - **N-gram Range:** (1, 1)
  - **Stop Words:** None
  - **Max Features:** 5,000
  - **Min DF:** 2, **Max DF:** 0.7
  - **Validation Accuracy:** 83.44%

- **Multinomial Naive Bayes**
  - **N-gram Range:** (1, 2)
  - **Stop Words:** English
  - **Max Features:** 5,000
  - **Min DF:** 5, **Max DF:** 0.9
  - **Validation Accuracy:** 84.32%

- **Decision Tree Classifier**
  - **N-gram Range:** (1, 1)
  - **Stop Words:** English
  - **Max Features:** 3,000
  - **Min DF:** 10, **Max DF:** 0.6
  - **Validation Accuracy:** 71.92%

The best-performing model on the validation set was the **Logistic Regression** model with a validation accuracy of 87.36%. This model was selected for final evaluation on the test set.

5.3 Best Model Selection and Test Set Evaluation

The Logistic Regression model achieved the highest validation accuracy of **87.36%**, making it the optimal model for our final evaluation on the test set.

Best Model: Logistic Regression
Validation Accuracy: 87.36%

We used the Logistic Regression model along with its vectorizer to transform the test data and evaluate performance on the test set. Below is the result:

Logistic Regression Test Accuracy: 85.22%

5.4 Analysis of Results

The Logistic Regression model performed best because it effectively captures relationships between words using bigrams and benefits from stop word removal. This approach allows it to detect subtle sentiment shifts, such as the difference between "not good" and "good." While other models, such as the SGD Classifier, also performed well, Logistic Regression achieved the highest accuracy with a balance between complexity and generalization.

The Random Forest and Decision Tree models, which rely on unigrams, had lower validation accuracies due to their limitations in capturing nuanced patterns in text data. Meanwhile, the Multinomial Naive Bayes model achieved good performance with bigrams, but its probabilistic approach was slightly less effective compared to the linear model of Logistic Regression.

5.5 Conclusion

This experiment demonstrates the importance of selecting appropriate n-gram configurations for different machine learning models. While Logistic Regression achieved the highest validation and test accuracy, the process of evaluating multiple models ensured that we selected the optimal model for the task. The final test accuracy of **85.22%** indicates that the chosen model generalizes well to unseen data.

6 Custom Naive Bayes Implementation

In this assignment, I implemented a custom Naive Bayes. This classifier uses a multinomial distribution to model word counts, making it well-suited for text classification tasks where features represent discrete frequency counts (e.g., occurrences of words in documents).

6.1 Why Multinomial Instead of Gaussian?

Multinomial Naive Bayes was chosen over Gaussian Naive Bayes because:

- **Feature Nature:** The dataset consists of text data where features are word or term frequencies, which align with the assumptions of a multinomial distribution.
- **Gaussian Limitation:** Gaussian Naive Bayes is more appropriate for continuous data and assumes the features are normally distributed, which does not fit the discrete nature of term counts.

- **Effective Smoothing:** The multinomial version applies Laplace (additive) smoothing to avoid zero probabilities for unseen words, which is critical for handling sparse text data.

6.2 Implementation Overview

- **Initialization:** The classifier takes a smoothing parameter α, which applies Laplacian smoothing to prevent zero probabilities.

- **Training (fit method):**

  - Computes the **prior probabilities** for each class:

$$P(\text{class}) = \frac{\text{count(class)}}{\text{total samples}}$$

  - Calculates the **log probabilities** of features given each class:

$$\log P(\text{feature}|\text{class}) = \log\left(\frac{\text{count(feature in class)} + \alpha}{\text{total feature count in class} + n \cdot \alpha}\right)$$

  - Stores these probabilities for making predictions.

- **Prediction (predict method):**

  - For each input sample, the classifier computes the posterior probability for all classes:

$$\log P(\text{class}|\text{features}) = \log P(\text{class}) + \sum(\text{feature counts} \times \log P(\text{feature}|\text{class}))$$

  - The class with the highest posterior probability is selected as the predicted label.

6.3 Evaluation Results

- **BoW + My Naive Bayes**

  - **Training Time:** 0.0114 seconds
  - **Prediction Time:** 0.1222 seconds
  - **Accuracy:** 0.8502
  - **Binary F1 Score:** 0.8410
  - **Macro-F1 Score:** 0.8497

BoW Model	Predicted Negative	Predicted Positive
Actual Negative	2044	206
Actual Positive	468	1782

Table 13: Confusion Matrix for BoW + My Naive Bayes

- **TF-IDF + My Naive Bayes**

  - **Training Time:** 0.0060 seconds
  - **Prediction Time:** 0.1195 seconds
  - **Accuracy:** 0.8709
  - **Binary F1 Score:** 0.8670
  - **Macro-F1 Score:** 0.8708

TF-IDF Model	Predicted Negative	Predicted Positive
Actual Negative	2026	224
Actual Positive	357	1893

Table 14: Confusion Matrix for TF-IDF + My Naive Bayes

- **One-Hot + My Naive Bayes**

  - **Training Time:** 0.0013 seconds
  - **Prediction Time:** 0.1149 seconds
  - **Accuracy:** 0.8578
  - **Binary F1 Score:** 0.8598
  - **Macro-F1 Score:** 0.8577

One-Hot Model	Predicted Negative	Predicted Positive
Actual Negative	1898	352
Actual Positive	288	1962

Table 15: Confusion Matrix for One-Hot + My Naive Bayes

6.4 Comparison with Previous Models

The results of the My Naive Bayes classifier are compared to the scikit-learn MultinomialNB implementation. Both implementations were evaluated on identical datasets with the same feature extraction methods—Bag-of-Words (BoW), TF-IDF, and One-Hot Encoding.

- **BoW Feature Comparison:** Both implementations achieved an accuracy of 85.02%, with nearly identical F1 scores (Binary F1: 0.8410, Macro-F1: 0.8497). This alignment highlights the robustness of both models when dealing with high-frequency word counts. It demonstrates that the custom implementation is reliable for basic frequency-based text classification.

  - **Old Accuracy (MultinomialNB):** 85.02%
  - **New Accuracy (My Naive Bayes):** 85.02%

- **TF-IDF Feature Comparison:** The custom implementation achieved 87.09% accuracy, matching the performance of the scikit-learn version. This shows that the My Naive Bayes model can effectively capture the importance of less frequent but meaningful words, underlining the strength of TF-IDF in combination with Naive Bayes models.
  - **Old Accuracy (MultinomialNB):** 87.09%
  - **New Accuracy (My Naive Bayes):** 87.09%

- **One-Hot Feature Comparison:** Both models performed similarly, with My Naive Bayes achieving 85.78% accuracy and a Binary F1 score of 0.8598. This suggests that the custom model can successfully handle categorical data, such as product titles or review summaries, without any loss in performance compared to the scikit-learn implementation.
  - **Old Accuracy (MultinomialNB):** 85.78%
  - **New Accuracy (My Naive Bayes):** 85.78%

6.5 Conclusion

The custom My Naive Bayes classifier closely mirrors the performance of the scikit-learn implementation, validating the correctness of the custom approach across different feature types. For Bag-of-Words, both models showed consistent results, confirming the simplicity and reliability of frequency-based features.

In the TF-IDF comparison, the custom model's ability to handle rare but significant words demonstrates its strength in more nuanced text classification tasks. Meanwhile, the One-Hot Encoding results reveal that My Naive Bayes can efficiently manage categorical data without sacrificing performance.

These results suggest that the custom classifier offers a viable alternative to the scikit-learn version and provides flexibility in a variety of text-processing scenarios.

7 References

References

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

7.1 Appendices

7.1.1 Class Distributions

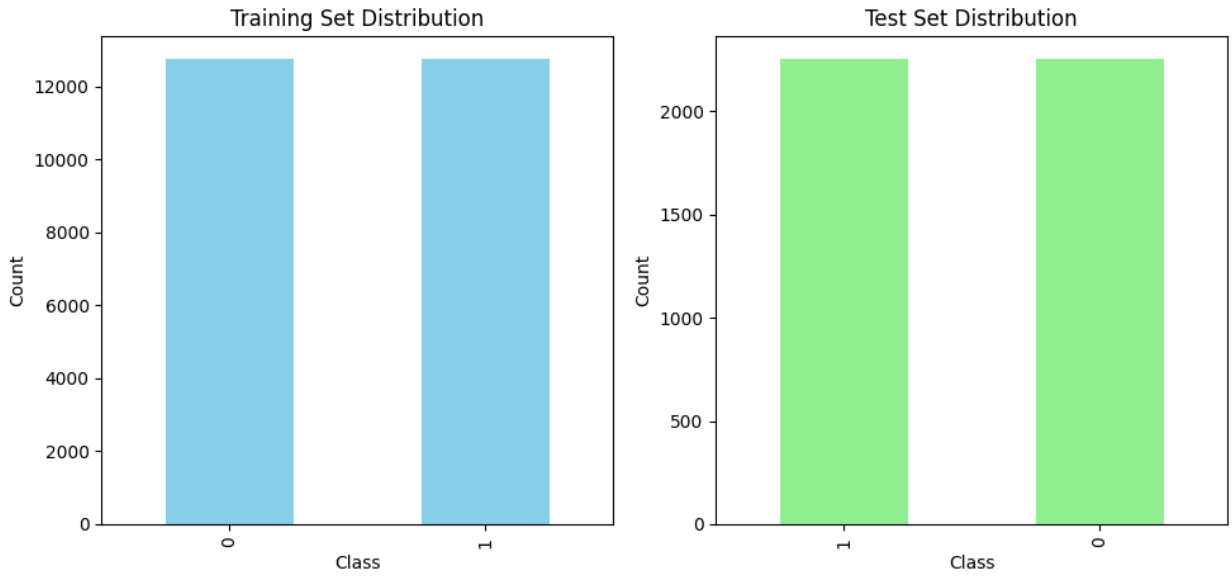

Figure 1: Class distributions in the training and test datasets.