NLP 220 ASSIGNMENT 1

Shriya Sravani Y
UCSC
sy4@ucsc.edu

1 Introduction

This report is divided into three parts, the first part focuses on implementing binary classification on a book review dataset from Amazon, using feature engineering techniques on Naive Bayes, SVM(Support Vector Machine) and Decision Tree classifiers. Nine such models are made with the aim to find the accuracies, macro F1 scores and their training and inference times to determine the best model.

The second part implements sentimental analysis on Stanford's movie review corpus. Five models are created to classify sentiment and report the accuracy for each of the models. The test accuracy of the 5 models is also reported.

The last part of this report shows the implementation of a custom Naive Bayes classifier and and gives a comparison between the custom and sklearn Naive Bayes used in the first part.

2 Part A - Feature Engineering for Naive Bayes, SVM and Decision Trees

2.1 Dataset

The dataset taken is a part of the Amazon Book reviews dataset. It contains reviews of books from Amazon and has 10 columns. The details of the dataset is given in Table 1.

Missing values in the columns of review/summary are filled to ensure proper data analysis. The column 'review/score' contains 5 values with equal distribution as shown in the Figure 1.

The review score of 4 or higher is set to be positive, and any score of 2 or less is set to be negative. Scores of 3 are ignored.

The dataset is divided into 85 % for training set and 15 % to test set. The distribution of the training and testing test is shown is Figure 2.

#	Column	Non-Null Count	Data Type
0	Id	37,500	object
1	Title	37,500	object
2	Price	5,285	float64
3	User_id	30,117	object
4	profileName	30,117	object
5	review/helpfulness	37,500	object
6	review/score	37,500	float64
7	review/time	37,500	int64
8	review/summary	37,495	object
9	review/text	37,500	object

Table 1: Summary of columns in the dataset

Figure 1: Distribution of Review Scores

2.2 Naive Bayes Classifier

The Naive Bayes classifier assumes that all features are independent of each other given the class label.For text classification, Multinomial Naive Bayes is commonly used, where it models the text as a distribution of word frequencies, making it suitable for analyzing word counts or term frequencies.

2.2.1 N-Grams

An n-gram is a contiguous sequence of 'n' items from a given text. Using n-grams as features allows machine learning models to capture more context

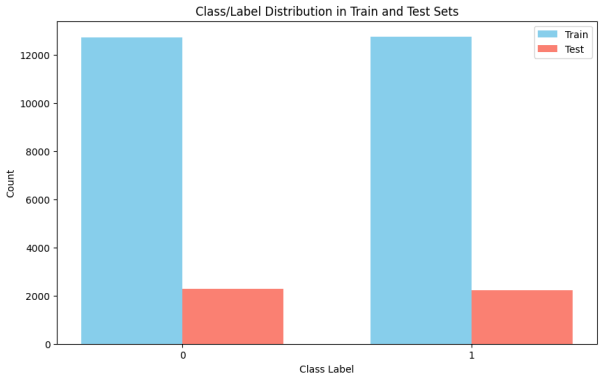

Figure 2: Distribution of train and test sets

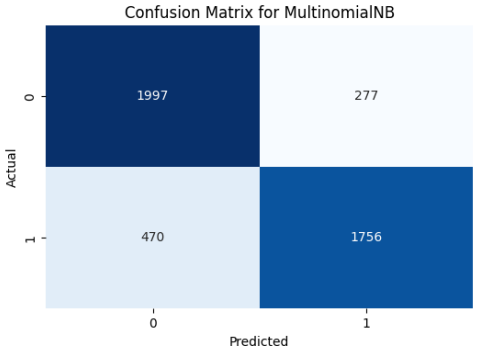

Figure 3: Confusion matrix

and relationships between words. In this code, a bigram (1, 2) model is used, meaning both individual words and consecutive word pairs are considered in the feature set, giving the model a richer understanding of the text data.

Feature Engineering

A CountVectorizer is initialized with ngram_range= (1, 2) and max_features=5000, meaning it captures unigrams and bigrams up to a maximum of 5000 features.

This vectorizer is then used to transform the training and test sets into n-gram feature matrices (X_train_vec and X_test_vec).

A Multinomial Naive Bayes classifier (MultinomialNB) is instantiated and trained on the n-gram features of the training data.

The evaluation function (evaluate_model) calculates the training time, inference time, accuracy, F1 score for each class, and macro F1 score. It also plots the confusion matrix for visual analysis.

Output

The model is slightly better at identifying negative reviews than positive reviews, as evidenced by a higher true negative count compared to true positives and slightly more false negatives.

Metric	Value
Accuracy	83.40%
F1 Score (Class 0 - Negative)	0.8424
F1 Score (Class 1 - Positive)	0.8246
Macro F1 Score	0.8335
Training Time	0.0231 seconds
Inference Time	0.0050 seconds

Table 2: Performance Summary of Multinomial Naive Bayes Classifier with N-Grams

The Multinomial Naive Bayes classifier with bigram features performs well on this dataset, with good accuracy and balanced class performance.

2.2.2 Bag-Of-Words

Bag of Words is a common text representation technique in natural language processing. In BOW, each document (e.g., a review) is represented by the frequency of each word in that document, disregarding grammar and word order. By converting text data into numerical vectors, BOW enables machine learning algorithms to interpret and learn from the data. In this code, we use a CountVectorizer to build a vocabulary of the 5,000 most frequent words in the dataset.

Feature Engineering

We use CountVectorizer with max_features=5000 to extract BOW features from the text data, which captures the most common 5,000 words. The feature vectors for both the training and test data are created (X_train_bow and X_test_bow).

A Multinomial Naive Bayes classifier (MultinomialNB) is trained on the BOW features of the training data.

The evaluation function evaluate_model calculates: Training time and inference time. Accuracy, F1 scores for each class, and the macro-average F1 score. A confusion matrix to visually analyze the model's classification performance.

Output

The model performs slightly better at identifying negative reviews than positive ones, though the performance is relatively balanced overall.

The Naive Bayes model trains quickly with BOW features due to the simplicity of both the BOW representation and the Naive Bayes algorithm.

Predictions are also fast, making Naive Bayes with BOW an efficient choice for text classification when real-time or large-scale predictions are required.

Metric	Value
Accuracy	83.38%
F1 Score (Class 0 - Negative)	0.8436
F1 Score (Class 1 - Positive)	0.8227
Macro F1 Score	0.8331
Training Time	0.0289 seconds
Inference Time	0.0043 seconds

Table 3: Performance Summary of Multinomial Naive Bayes Classifier with Bag of Words

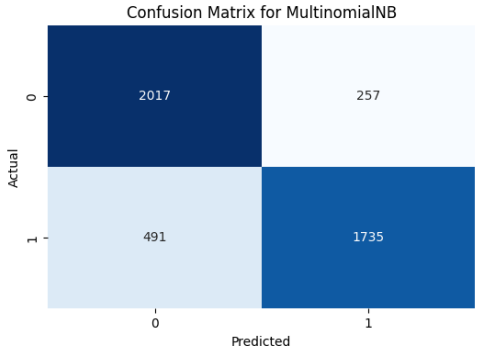

Figure 4: Confusion matrix

2.2.3 TF-IDF

TF-IDF is a text representation technique used to convert textual data into numerical vectors for machine learning. It calculates the importance of a word in a document relative to its frequency across the entire dataset. The TF (Term Frequency) measures how frequently a word appears in a document, while IDF (Inverse Document Frequency) downscales words that occur frequently across many documents, as they are often less informative.

Feature Engineering

A TfidfVectorizer is initialized with max_features=5000, meaning it selects the 5,000 most important terms based on TF-IDF values. The vectorizer transforms both the training and test sets into TF-IDF feature matrices (X_train_tfidf and X_test_tfidf). The evaluation function, evaluate_model, calculates the following: -Training and inference times to measure model efficiency. -Accuracy, F1 scores for each class, and the macro-average F1 score. -A confusion matrix to visualize model performance for each class.

Output

The Multinomial Naive Bayes model trains very quickly with TF-IDF features, benefiting from both the simplicity of Naive Bayes and the compact representation of TF-IDF.

Predictions are also extremely fast, making this approach suitable for real-time or large-scale applications.

Metric	Value
Accuracy	85.18%
F1 Score (Class 0 - Negative)	0.8541
F1 Score (Class 1 - Positive)	0.8494
Macro F1 Score	0.8517
Training Time	0.0206 seconds
Inference Time	0.0025 seconds

Table 4: Performance Summary of Multinomial Naive Bayes Classifier with TF-IDF

The confusion matrix shows that the model performs well on both classes, with fewer misclassifications compared to simpler representations like Bag of Words.

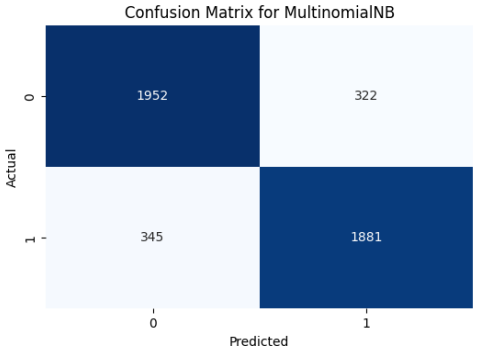

Figure 5: Confusion matrix

2.3 Support Vector Machine(SVM)

SVM is a powerful classification algorithm used in text and other high-dimensional data classification. It works by finding a hyperplane that best separates the data points into different classes. SVC (Support Vector Classifier), a type of SVM, is particularly effective in handling complex datasets where classes are not linearly separable.

2.3.1 N-Grams

In this model, bi-grams (1-gram and 2-gram combinations) are used to capture both single words and pairs of consecutive words, enriching the feature set for the classifier.

Feature Engineering

The CountVectorizer with ngram_range=(1, 2) is used to extract features from the text data. It generates both unigrams and bigrams, with a maximum of 5,000 features. These n-gram features are used to transform the training and testing datasets into sparse matrices (X_train_vec and X_test_vec).

Output

The model takes extensive time for training and inference. SVM is computationally expensive, especially with high-dimensional n-gram features, making training slower compared to simpler models like Naive Bayes. SVM's prediction time is also longer due to the complex calculations involved.

Metric	Value
Accuracy	85.51%
F1 Score (Class 0 - Negative)	0.8546
F1 Score (Class 1 - Positive)	0.8556
Macro F1 Score	0.8551
Training Time	574.5870 seconds
Inference Time	70.5694 seconds

Table 5: Performance Summary of SVM Classifier with N-grams (Bi-grams)

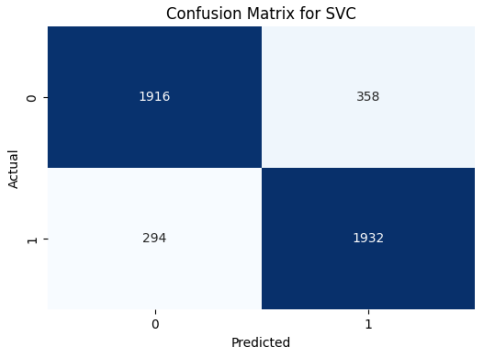

Figure 6: Confusion matrix

2.3.2 Bag-Of-Words

BOW converts text into numerical features based on the occurrence of words within a document. Each unique word in the dataset forms a feature, and the values represent the count of occurrences of each word in the document. SVM can handle both linear and non-linear decision boundaries, but in this example, we are using SVC, which uses the default radial basis function (RBF) kernel.

Feature Engineering

CountVectorizer is initialized with max_features=5000, restricting the vocabulary to the 5,000 most common words in the dataset.

This vectorizer transforms both the training and test sets into a sparse matrix of word counts (X_train_bow and X_test_bow).

Output

The F1 score of 0.8439 for class 0 (negative) and 0.8458 for class 1 (positive), indicating balanced performance across both classes.

The SVM model requires significant time to train on the BOW feature set, as SVM is computationally intensive, especially with high-dimensional features like word counts.

Similarly, prediction time is high due to the complexity of SVM, which needs to calculate distances for each support vector in the model to make predictions.

Metric	Value
Accuracy	84.49%
F1 Score (Class 0 - Negative)	0.8439
F1 Score (Class 1 - Positive)	0.8458
Macro F1 Score	0.8449
Training Time	390.8133 seconds
Inference Time	54.6385 seconds

Table 6: Performance Summary of SVM Classifier with Bag of Words

The confusion matrix shows that the model is balanced in its predictions for both classes, with a slight tendency toward false positives, meaning it occasionally classifies negative reviews as positive.

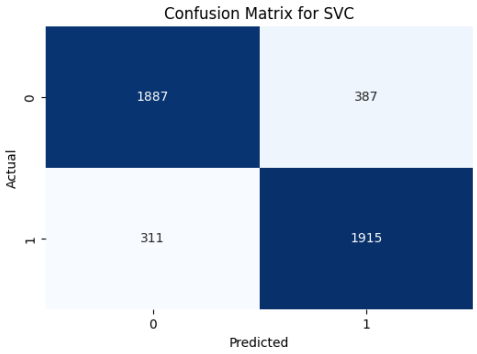

Figure 7: Confusion matrix

2.3.3 TF-IDF

The Term Frequency (TF) component captures the frequency of a word in a document, while the Inverse Document Frequency (IDF) component downscales commonly occurring words across documents, giving more weight to unique or informative words. SVM is powerful for text classification because it can handle high-dimensional feature spaces, such as those created by TF-IDF, to create accurate and reliable decision boundaries.

Feature Engineering

We initialize TfidfVectorizer with max_features=5000, which restricts the vocabulary to the 5,000 most informative words based on TF-IDF scores.

This vectorizer then transforms the training and test sets into TF-IDF feature matrices (X_train_tfidf and X_test_tfidf), creating a numerical representation of the text data.

SVM with TF-IDF features is computationally intensive during training, due to the high-dimensional feature space created by TF-IDF and the SVM's optimization process for finding the optimal hyperplane.

Metric	Value
Accuracy	88.53%
F1 Score (Class 0 - Negative)	0.8861
F1 Score (Class 1 - Positive)	0.8846
Macro F1 Score	0.8853
Training Time	718.4425 seconds
Inference Time	49.7045 seconds

Table 7: Performance Summary of SVM Classifier with TF-IDF Features

The confusion matrix shows that the model is slightly better at predicting positive labels than negative ones. Both classes show high true positive counts, indicating strong predictive power.

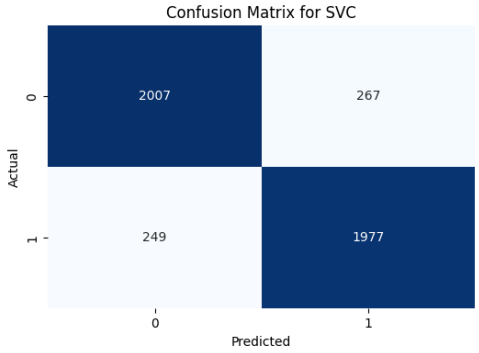

Figure 8: Confusion matrix

2.4 Decision Trees

Decision Trees are supervised learning models used for classification tasks. They work by recursively partitioning the dataset into subsets based on feature values, aiming to maximize the purity (homogeneity) of each subset at each split. Each internal node in a decision tree represents a decision based on a feature, while leaf nodes represent class labels.

2.4.1 N-Grams

Feature Engineering

The CountVectorizer is set with ngram_range=(1, 2) and max_features=5000, capturing both unigrams and bigrams with a vocabulary size of up to 5,000 words. This vectorizer transforms the training and test data into a matrix of n-gram counts (X_train_vec and X_test_vec), creating a numerical representation for the Decision Tree classifier.

Output

The Decision Tree model is relatively fast to train compared to more complex algorithms like SVM, especially on n-gram features. Decision Trees have efficient prediction times since once the tree is built, classifying new samples involves simple tree traversal.

Metric	Value
Accuracy	71.09%
F1 Score (Class 0 - Negative)	0.7118
F1 Score (Class 1 - Positive)	0.7099
Macro F1 Score	0.7109
Training Time	39.7355 seconds
Inference Time	0.0051 seconds

Table 8: Performance Summary of Decision Tree Classifier with N-grams (Bi-grams)

The confusion matrix shows that the model struggles with both classes, with relatively high false positives and false negatives. Decision Trees struggles with high-dimensional text features.

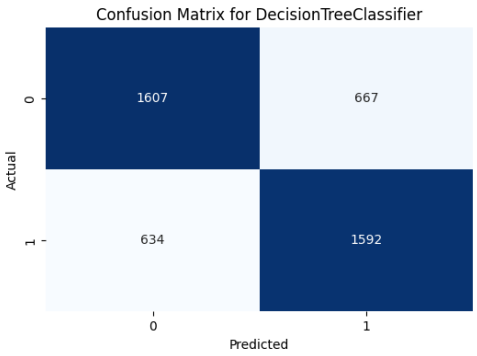

Figure 9: Confusion matrix

2.4.2 Bag-Of-Words

Feature Engineering

CountVectorizer is initialized with max_features=5000, which restricts the vocabulary to the 5,000 most frequent words. This vectorizer transforms both the training and test data into a sparse matrix of word counts (X_train_bow and X_test_bow), creating a numerical representation of the text data.

Output

The macro F1 score shows that the DT overall performance is moderate.

The Decision Tree model is relatively fast to train, even with high-dimensional BOW features.

Predictions are also fast due to the efficient tree traversal process used by Decision Trees.

Metric	Value
Accuracy	71.20%
F1 Score (Class 0 - Negative)	0.7119
F1 Score (Class 1 - Positive)	0.7121
Macro F1 Score	0.7120
Training Time	28.0755 seconds
Inference Time	0.0042 seconds

Table 9: Performance Summary of Decision Tree Classifier with Bag of Words

The confusion matrix indicates that the Decision Tree model has moderate performance and struggles with both classes, as reflected by relatively high numbers of false positives and false negatives.

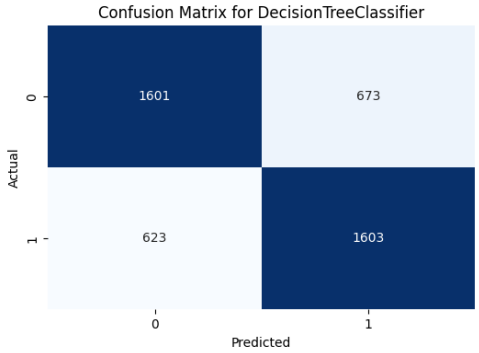

Figure 10: Confusion matrix

2.4.3 TF-IDF

Feature Engineering

TfidfVectorizer with max_features=5000 is used to transform the text data into TF-IDF features, restricting the vocabulary to the 5,000 most important terms. This vectorizer creates the TF-IDF feature matrices (X_train_tfidf and X_test_tfidf) for both training and test sets, providing a weighted representation of the text.

Output

The model displays moderate performance, showing it may have limitations in capturing complex patterns with TF-IDF features.

The Decision Tree model is efficient to train, even with complex TF-IDF features.

Predictions are fast because Decision Trees require minimal computation to traverse the tree for classification.

The confusion matrix indicates moderate performance, with substantial numbers of false positives and false negatives. This is common with high-dimensional representations like TF-IDF when used with Decision Trees, as they can capture noise or irrelevant details.

Metric	Value
Accuracy	70.87%
F1 Score (Class 0 - Negative)	0.7076
F1 Score (Class 1 - Positive)	0.7098
Macro F1 Score	0.7087
Training Time	37.0094 seconds
Inference Time	0.0044 seconds

Table 10: Performance Summary of Decision Tree Classifier with TF-IDF Features

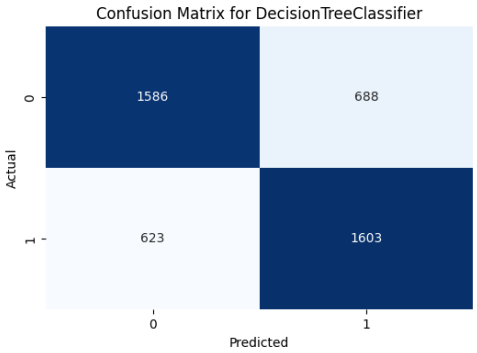

Figure 11: Confusion matrix

In conclusion, the SVM with Tf-IDf gave the highest accuracy. The n-grams model with Tf-idf took the least training and inference time. The decision tree model with the n-gram underperformed due to noise and the SVM model with Tf-Idf took the most time to train and infer. Overall, the N-grams model bag-of-words performed well.

3 Part B - Sentiment Analysis on Stanford's total movie review corpus

3.1 Dataset

The Stanford Large Movie Review Dataset (IMDb) contains 50,000 movie reviews labeled for sentiment analysis. The dataset is balanced, with 25,000 positive reviews and 25,000 negative reviews. It is divided into two main splits:

Train split (25,000 reviews): Used for model training and hyperparameter tuning.

Test split (25,000 reviews): Used for final evaluation.

Each split is organized into "pos" and "neg" folders for positive and negative sentiment labels, respectively. The train set was further split into 90% for training and 10% for validation, enabling model

tuning and performance assessment on unseen data before the final test evaluation.

The dataset is parsed by reading each review file from the "pos" and "neg" folders in the train and test splits, and the labels are assigned as 1 for positive and 0 for negative reviews. The parsed data is saved in a CSV format (imdb_reviews.csv) for easier processing.

Figure 12: Confusion matrix

Feature Engineering

Feature Extraction(N-Grams)

N-grams with ranges of (1,2) (unigrams and bigrams) are extracted using CountVectorizer with stopwords removed and min_df=3 to exclude infrequent terms. The top 5,000 features are retained, providing a compact yet informative representation of the text for classification.

Classifiers Used:

- Naive Bayes: A probabilistic classifier suitable for text data, assuming feature independence.

- Logistic Regression: A linear model using a sigmoid function to output class probabilities.

- Random Forest: An ensemble model that builds multiple decision trees and aggregates their predictions.

- Decision Tree: A tree-based model that makes sequential splits to classify data points.

- Support Vector Machine (SVM): A powerful classifier that separates classes by maximizing the margin around a decision boundary.

Model Training and Evaluation:

Each model is trained on the training data, and its performance is evaluated on the validation and test sets. Accuracy is calculated for each model on both validation and test splits to assess generalization.

Output

Logistic Regression achieved the highest accuracy on both validation (87.48%) and test (87.10%) sets, making it the best-performing model in this setup.

Naive Bayes and SVM also performed well, achieving test accuracies of 83.79% and 86.82%, respectively.

Decision Tree showed the lowest accuracy (71.26%), indicating sensitivity to noise in the text features.

Model	**Validation Accuracy**	**Test Accuracy**
Naive Bayes	83.80%	83.79%
Logistic Regression	87.48%	87.10%
Random Forest	84.60%	84.24%
Decision Tree	71.28%	71.26%
SVM	86.76%	86.82%

Table 11: Model Validation and Test Accuracies

The best test set accuracy is of logistic regression.

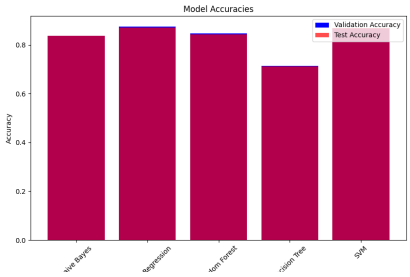

Figure 13: Model Accuracy

4 Part C- Custom Naive Bayes Implementation

Implementing a custom Naive Bayes classifier and comparing its performance with Scikit-Learn's MultinomialNB across three different feature extraction methods: Bag of Words (BOW), n-grams, and TF-IDF.

Feature Engineering

The code first preprocessed the data using three feature extraction methods (BOW, n-grams, and TF-IDF), and then trained both the custom Naive Bayes and Scikit-Learn's Naive Bayes model on each of these representations. The results were evaluated using accuracy, macro F1 score, and a confusion matrix to assess the classification performance for both models.

Feature Extraction	Model	Accuracy	Macro F1 Score	Confusion Matrix
BOW	Custom Naive Bayes	0.8422	0.8421	{[1948, 326], [384, 1842]}
	Scikit-Learn Naive Bayes	0.8338	0.8331	{[1948, 326], [384, 1842]}
n-grams	Custom Naive Bayes	0.7918	0.7918	{[1776, 498], [439, 1787]}
	Scikit-Learn Naive Bayes	0.8340	0.8335	{[1776, 498], [439, 1787]}
TF-IDF	Custom Naive Bayes	0.8422	0.8421	{[1948, 326], [384, 1842]}
	Scikit-Learn Naive Bayes	0.8518	0.8517	{[1948, 326], [384, 1842]}

Table 12: Comparison of Custom Naive Bayes and Scikit-Learn Naive Bayes with Different Feature Extraction Methods

- **Bag of Words (BOW)**:
  - Custom Naive Bayes performed marginally better on BOW features, likely due to the way feature probabilities are handled in the custom implementation.
- **N-grams**:
  - Scikit-Learn's model performed better on n-grams, possibly because it has optimized handling of sparse features in high-dimensional spaces, which is common in n-gram representation.
- **TF-IDF**:
  - Scikit-Learn's **MultinomialNB** outperformed the custom model with an accuracy of **85.18%** and a macro F1 score of **0.8517**.

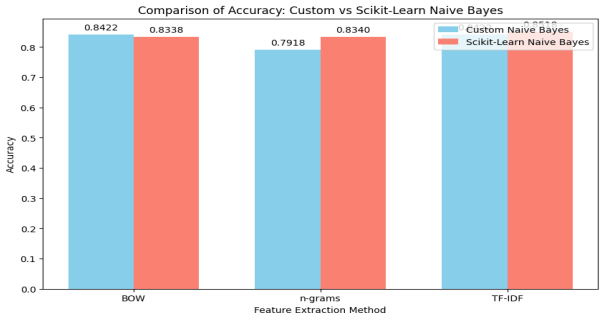

Figure 14: Model Accuracy

5 Conclusion

This report details experiments conducted for binary sentiment classification using feature engineering and model comparison across Naive Bayes, SVM, and Decision Tree classifiers on Amazon Book Reviews and Stanford Movie Review datasets.