Sentence Classification using Machine Learning Models

Asmae Mouradi
Department of Computing
Wichita State University
Wichita, United States of America
axmouradi@shockers.wichita.edu

Shruti Kshirsagar
Department of Computing
Wichita State University
Wichita, United States of America
shruti.kshirsagar@wichita.edu

***Abstract*—Sentence classification is a core task in Natural Language Processing (NLP), aimed at categorizing text based on predefined labels. This project addresses the classification of Amazon product reviews into positive or negative categories using machine learning models. The pipeline follows key steps, including data preprocessing, feature extraction via TF-IDF (Term Frequency-Inverse Document Frequency), and model training with three distinct algorithms: Random Forest, Decision Tree, and XGBoost. The models are evaluated on a variety of metrics such as accuracy, precision, recall, and F1-score. Experimental results demonstrate that XGBoost, an ensemble learning method, outperforms the simpler models, showing its suitability for sentence classification tasks where interpretability and performance are critical.**

***Index Terms*—Machine learning pipeline, XGBoost, Random Forest, Decision Tree, Sentence Classification.**

I. Introduction

The primary objective of this project is to develop a model that classifies Amazon product reviews into two categories: positive and negative. The classification will be performed using three machine learning models: Random Forest, Decision Tree, and XGBoost. The project follows a structured machine learning pipeline, starting from data preprocessing to feature extraction using TF-IDF (Term Frequency-Inverse Document Frequency). The models will be evaluated based on key metrics such as accuracy, precision, recall, and F1-score. The goal is to assess which model provides the best performance for sentiment classification in Amazon reviews.

Sentence classification plays a pivotal role in numerous NLP applications, from document categorization to sentiment analysis. In this project, we focus on sentence classification using the Amazon review dataset, where the goal is to categorize reviews as either positive or negative. The task requires effective feature extraction, model training, and evaluation to determine the most appropriate approach for real-world sentiment classification tasks.

The choice of Random Forest, Decision Tree, and XGBoost for this project is motivated by their robustness in handling classification tasks with textual data. While Decision Trees provide transparency and simplicity in model interpretation, they are prone to overfitting, especially on complex data. Random Forests, an ensemble of decision trees, mitigate this problem by averaging multiple decision trees, improving generalization and reducing variance. However, XGBoost—a more advanced form of gradient boosting—has gained significant attention for its high performance in classification tasks. XGBoost is known for its ability to handle large, imbalanced datasets and its regularization techniques, which reduce overfitting and boost predictive accuracy. These characteristics make XGBoost particularly effective for the sentence classification task in this project.

Furthermore, hyperparameter tuning plays a critical role in maximizing the performance of these models. While the base models provide a solid foundation, fine-tuning hyperparameters such as the number of trees, learning rate, and tree depth can significantly enhance model performance. In this work, we employ techniques like GridSearchCV and RandomizedSearchCV to explore the hyperparameter space for XGBoost, Random Forest, and Decision Tree models. By identifying the optimal configuration for each model, we aim to improve their predictive capabilities and compare their performance on Amazon product reviews.

This combination of carefully selected models and systematic hyperparameter tuning offers a comprehensive approach to sentence classification.

The remainder of this paper is organized as follows. In Section II Related Work, in Section III we describe our methodology, including the deep learning models, data collection and preprocessing steps, model architecture and hyperparameter tuning. Section IV presents the dataset and its exploratory analysis through descriptive statistics and visualizations. In Section V we detail the experimental setup, training and evaluation metrics, and report the results of model training and performance comparisons. Section VI offers an in-depth discussion of these results, the challenges we encountered, and how our findings relate to existing work. Finally, Section VII summarizes the key findings, discusses limitations, and outlines directions for future research.

II. Related Work

Convolutional Neural Networks (CNNs) have proven highly effective in automating classification tasks across diverse domains, including computer vision and natural language processing (NLP). With the growing volume of digital text, automatically categorizing large datasets has become essential, particularly for businesses that rely on customer feedback. This is especially true for tasks like sentiment analysis of Amazon reviews, where understanding sentiment can influence strategic decisions.

Previous research, notably by Kim [1], demonstrated the potential of CNNs for sentence classification using pre-trained word embeddings such as Word2Vec. Kim's work explored various configurations of CNN architectures, such as static, non-static, and multi-channel, to improve performance through fine-tuning. While CNNs have shown success in text classification tasks, they generally require large labeled datasets, which limits their applicability in settings where labeled data is scarce.

This paper [2] surveys various machine learning models used for NLP tasks, including sentence classification. It discusses the application of ensemble models like XGBoost and Random Forest in text classification tasks and compares their performance to other methods.

This survey paper [3] discusses the recent advances in deep learning models for text classification tasks. While it primarily focuses on deep learning, it also provides comparisons with traditional machine learning methods like Random Forest, Decision Trees, and XGBoost.

This seminal paper [4] introduces XGBoost, an ensemble method that has become widely popular in many machine learning tasks, including sentence classification. The paper details the efficiency and scalability of XGBoost and discusses its application in structured and unstructured data classification tasks, including text classification.

This paper [5] provides a comprehensive overview of various deep learning techniques used for text classification, including CNNs and RNNs, and contrasts them with classical machine learning techniques such as Decision Trees and Random Forests.

This paper [6] evaluates the impact of various feature engineering techniques like TF-IDF and others for text classification. It also compares the performance of XGBoost and other models such as Random Forest and SVM.

This foundational paper [7] introduces Random Forests and discusses their use in high-dimensional datasets like text classification. Although it's older, it remains an important reference for traditional classification techniques, especially for comparison with newer models like XGBoost.

This project, uses traditional machine learning models like XGBoost, Random Forest, and Decision Trees for sentence classification tasks. These models offer significant advantages in terms of interpretability and computational efficiency, especially when datasets are smaller or computational resources are limited. However, the importance of CNN-based approaches, as demonstrated by Kim, can still be acknowledged as a foundational method in the broader landscape of sentence classification, especially when deep learning techniques become more feasible.

III. Methodology

A. Detailed Explanation of the Machine Learning Model(s) Used

In this project, three machine learning models were employed for sentiment classification of Amazon product reviews: Random Forest, Decision Tree, and XGBoost. Each model was chosen for its distinct advantages in handling classification tasks.

1) Random Forest Classifier: Random Forest is an ensemble learning method based on decision trees. It operates by constructing a multitude of decision trees during training and outputs the class that is the mode of the classes of the individual trees. The model uses a technique called bootstrapping (sampling with replacement) to create multiple subsets of the data, and each decision tree is trained on a different subset.

Key Characteristics:

- Combines multiple weak learners (decision trees) to create a strong learner.
- Each tree is trained on a random subset of the data, which reduces variance and prevents overfitting.
- During the construction of each tree, only a random subset of features is considered for splitting at each node, further increasing model robustness.
- Random Forest can estimate its performance by evaluating the samples that were not included in the bootstrap sample.

In this project, Random Forest was used due to its robustness against overfitting, especially with high-dimensional data like text.

2) Decision Tree Classifier: Decision Trees are one of the simplest machine learning algorithms used for classification. It works by recursively splitting the data based on feature values to create a tree-like structure of decisions. The goal is to partition the data into homogenous subsets with respect to the target variable sentiment in my project.

Key Characteristics:

- Decision Trees are easy to understand and interpret, providing a clear decision-making process.
- It uses a top-down approach to split the data based on the best feature at each step, usually by maximizing information gain or minimizing Gini impurity.

- Decision Trees are prone to overfitting, especially when the tree is allowed to grow deep. Pruning techniques like setting a maximum depth are often used to combat this.

The Decision Tree model was selected for its interpretability.

3) XGBoost (Extreme Gradient Boosting): XGBoost is an implementation of gradient boosting that has gained popularity due to its superior performance in many machine learning tasks. It builds an ensemble of weak learners (decision trees), where each tree corrects the errors made by the previous ones. XGBoost is particularly known for its efficiency, speed, and scalability, making it suitable for large datasets.

Key Characteristics:

- It uses the gradient descent optimization method to minimize a loss function, iteratively adding trees to correct the errors of the existing model.
- XGBoost includes L1 and L2 regularization, which helps to control overfitting and makes the model more robust.
- XGBoost can handle missing values naturally, using a process called "sparse-aware" learning.
- XGBoost supports parallel and distributed computing, making it highly efficient for large datasets.

XGBoost was chosen for its ability to handle complex datasets and deliver high performance, particularly in tasks like sentiment classification where model accuracy and efficiency are crucial.

B. Data Collection and Preprocessing Techniques

The dataset used is the Amazon Reviews dataset, which consists of 3,600,000 rows and 3 columns: `sentiment`, `review_title`, and `review_text`. The `sentiment` column represents the sentiment label associated with each review, where the value 1 indicates negative sentiment and the value 2 indicates positive sentiment. For the purpose of binary classification, these labels were remapped so that 1 becomes 0 (negative) and 2 becomes 1 (positive).

Data preprocessing was critical to ensure that the raw text data was cleaned, normalized, and transformed into a format suitable for model training. The preprocessing pipeline consisted of the following steps:

1) **Handling Missing Values:** Missing entries in the dataset, particularly in the `review_title` column, were addressed by filling them with a placeholder (e.g., "No title") to maintain data consistency without losing valuable information.
2) **Text Cleaning:** The raw review text was converted to lowercase and cleaned by removing special characters, punctuation, and extra whitespace. Tokenization was then applied to split the text into individual words, and stopwords were removed to reduce noise in the dataset.
3) **Outlier Removal:** the Interquartile Range (IQR), were used to detect and remove outliers in review lengths. Reviews that contained an unusually high number of words were filtered out to ensure that the training data was uniform and to minimize the impact of noisy data.
4) **Feature Extraction:** The cleaned text data was transformed into numerical features using Term Frequency-Inverse Document Frequency (TF-IDF) vectorization. This method captures the importance of words within each review relative to the entire dataset.
5) **Data Splitting:** Finally, the dataset was split into training, validation, and testing subsets using stratified sampling. This approach ensured that the distribution of sentiment labels remained consistent across all data splits.

The preprocessing techniques helped to prepare the dataset for efficient model training and evaluation by ensuring that the input data was both clean and representative of the overall sentiment distribution in the Amazon Reviews dataset.

C. Model Architecture and Parameters

Model	Description	Hyperparameters Tried	Final Values
Random Forest	Ensemble of decision trees that reduces variance by averaging.	• n_estimators: 100–300 • max_depth: None, 20, 40 • min_samples_split: 2–10 • min_samples_leaf: 1–10 • bootstrap: True/False	• n_estimators=250 • max_depth=None • min_samples_split=4 • min_samples_leaf=1 • bootstrap=True
Decision Tree	Single tree splitting on features to minimize impurity.	• criterion: gini/entropy • max_depth: None, 20, 40 • min_samples_split: 2–10 • min_samples_leaf: 1–10	• criterion=entropy • max_depth=None • min_samples_split=3 • min_samples_leaf=2
XGBoost	Gradient-boosted trees with regularization to control overfitting.	• learning_rate: 0.01–0.1 • n_estimators: 100–300 • max_depth: 3–8 • subsample: 0.7–1.0 • colsample_bytree: 0.7–1.0	• learning_rate=0.05 • n_estimators=200 • max_depth=5 • subsample=0.8 • colsample_bytree=0.8

TABLE I
MODEL DESCRIPTIONS, HYPERPARAMETERS SEARCHED, AND FINAL CHOSEN VALUES.

D. Training Process and Hyperparameter Tuning

First, the entire end-to-end pipeline is executed via four scripts:

1) **Exploratory Data Analysis**
   `$ python eda.py`
   - Loads `data/train.csv` and `data/test.csv`.
   - Generates distribution, missing-value, and outlier plots in `./figures`.
2) **Preprocessing (Clean & Split)**
   `$ python preprocessing.py`
   - Reads raw `train.csv` and `test.csv`.
   - Fills nulls, removes outliers ($1.5 \times$ IQR), cleans text (lowercase, remove non-words, tokenize, drop stop words).
   - Saves cleaned files as `train_cleaned_final.csv` and `test_cleaned_final.csv`.
3) **Modeling (Train & Tune)**
   `$ python modeling.py`
   - Loads cleaned CSVs, vectorizes via TF–IDF (unigrams + bigrams).
   - Runs `RandomizedSearchCV` (3-fold CV) over predefined parameter distributions for RF, DT, and XGB.
   - Compares via cross-validation and builds a stacking ensemble (Logistic Regression meta-learner).
   - Saves tuned models and TF–IDF vectorizer in `./models`.
4) **Evaluation (Test & Visualize)**
   `$ python evaluation.py`
   - Loads cleaned test set and trained models.
   - Computes accuracy, precision, recall, F1-score and classification reports.
   - Generates & saves confusion matrices in `./confusion_matrices`.

Next, we focus on the actual hyperparameter searches:

a) Random Forest:

- *Search space:* `n_estimators` = [100,300], `max_depth` $\in \{\text{None}, 20, 40\}$, `min_samples_split` = [2,10], `min_samples_leaf` = [1,10], `bootstrap` $\in \{\text{True}, \text{False}\}$
- *Best found:* {`'n_estimators':181, 'max_depth':20, 'min_samples_split':2, 'min_samples_leaf':1, 'bootstrap':True`}

b) Decision Tree:

- *Search space:* `criterion` $\in \{\text{gini}, \text{entropy}\}$, `max_depth` $\in \{\text{None}, 20, 40\}$, `min_samples_split` = [2,10], `min_samples_leaf` = [1,10]
- *Best found:* {`'criterion':'gini', 'max_depth':20, 'min_samples_split':2, 'min_samples_leaf':2`}

c) XGBoost:

- *Search space:* `learning_rate` $\sim \mathcal{U}(0.01, 0.1)$, `n_estimators` = [100,300], `max_depth` = [3,8], `subsample` $\sim \mathcal{U}(0.7, 1.0)$, `colsample_bytree` $\sim \mathcal{U}(0.7, 1.0)$
- *Best found:* {`'learning_rate':0.0975, 'n_estimators':171, 'max_depth':5,`

```
'subsample':0.9197,
'colsample_bytree':0.8749}
```

IV. Data

A. Description of the Dataset(s)

The dataset used in this project is the **Amazon Reviews** dataset, which consists of customer reviews collected from the Amazon platform. It contains a total of 3,600,000 rows and 3 columns: `sentiment`, `review_title`, and `review_text`. The `sentiment` column represents the label associated with each review, where the value 1 indicates a negative sentiment and the value 2 indicates a positive sentiment. For the purpose of binary classification, label 1 is mapped to 0 (negative) and label 2 is mapped to 1 (positive). The `review_title` column provides a short title summarizing the review, while the `review_text` column contains the full text of the customer review.

B. Data Exploration and Visualization

In this section, we perform a thorough exploratory data analysis (EDA) on the raw Amazon Reviews dataset to understand its structure and quality before any modeling. We begin by summarizing the overall sentiment distribution, then inspect missing values and duplicate entries. Next, we compute and visualize the distribution of review lengths using histograms and box plots—to detect potential outliers. All key findings are illustrated with visualizations (pie charts, bar plots, histograms, and box plots) which are saved under the `./figures` directory for easy reference.

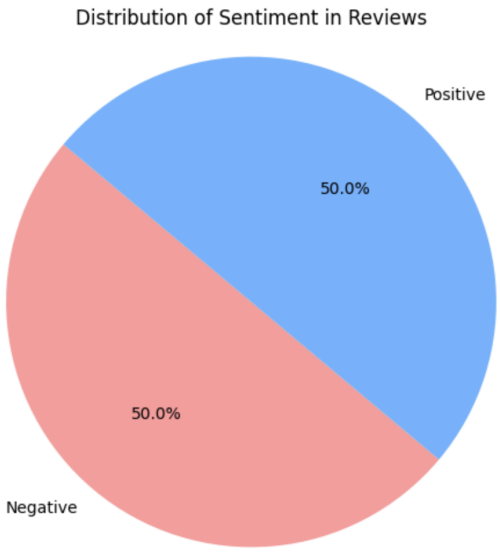

Fig. 1. Sentiment Distribution in the Amazon Reviews Dataset

Figure 1 shows the distribution of sentiment labels in the Amazon Reviews dataset. The dataset is balanced, with an approximately equal number of positive (label 2) and negative (label 1) reviews. Each sentiment class contains around 1.8 million instances, helping ensure that the model does not develop bias toward either class.

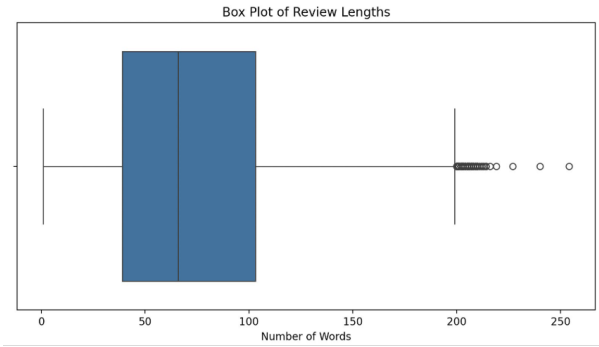

Fig. 2. Box Plot of Review Lengths Showing Outliers in the Dataset

Figure 2 presents a box plot illustrating the distribution of review lengths in the dataset. Most reviews contain between 20 and 150 words, indicating a fairly consistent range of text lengths. However, a small number of reviews exceed 200 words and are identified as potential outliers. A total of 537 such outliers were detected and removed to maintain uniformity and reduce noise in the training data.

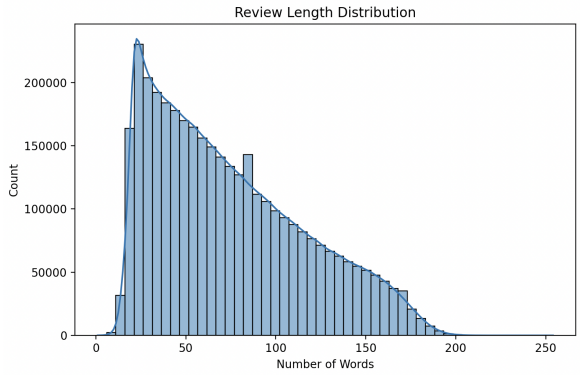

Fig. 3. Histogram of Review Lengths in the Amazon Reviews Dataset

Figure 3 illustrates the distribution of review lengths in the dataset. Most reviews are relatively short, with a large concentration between 20 and 100 words. The distribution follows a right-skewed pattern, indicating that while shorter reviews are more frequent, a smaller number of longer reviews extend beyond 150 words. This visualization supports the decision to filter out extremely long reviews as outliers in the preprocessing stage.

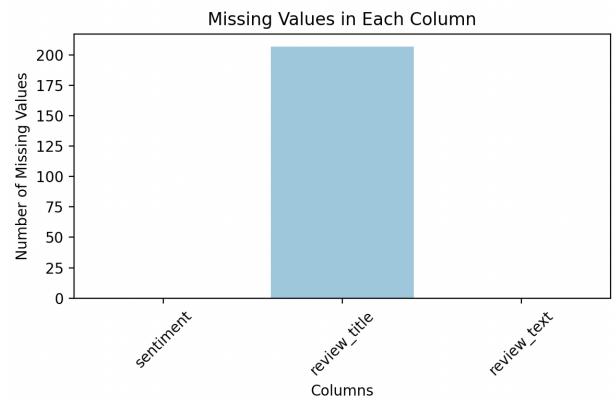

Fig. 4. Missing Values in Each Column of the Amazon Reviews Dataset

Figure 4 displays the number of missing values per column in the dataset. While the `sentiment` and `review_text` columns are complete, the `review_title` column contains 207 missing entries. Since the title is not critical for sentiment prediction, these missing values can either be ignored or filled with a placeholder during preprocessing.

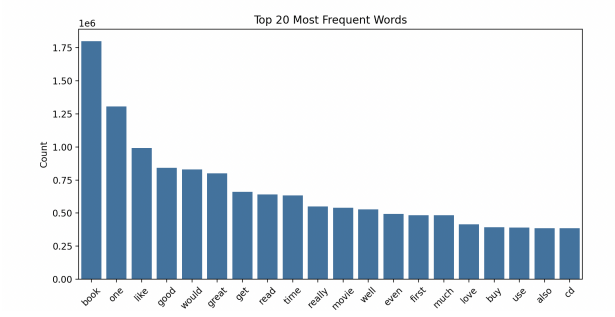

Fig. 5. Top 20 most Frequent Words

Figure 5 presents the top 20 most frequent words in the dataset after text cleaning. The term *"book"* dominates the frequency distribution, followed by commonly used review-related words such as *"one"*, *"like"*, *"good"*, and *"great"*.

V. Experiments and Results

A. Presentation of Experimental Setup

The dataset used for this experiment consists of textual reviews along with labels indicating whether the sentiment of the review is positive or negative.

The preprocessed dataset is divided into a training set and a testing set, with 80% of the data used for training and 20% reserved for testing. The text data is transformed into numerical features using the Term Frequency-Inverse Document Frequency (TF-IDF) method, which captures the importance of words in the context of each review.

Three machine learning models are implemented for sentiment classification:

- **Random Forest**: An ensemble method that uses multiple decision trees to reduce variance by averaging their predictions.
- **Decision Tree**: A model that splits data into subsets based on feature values to minimize impurity at each split.
- **XGBoost**: A gradient-boosting model known for its high performance and scalability, especially in large datasets.

The models are initially evaluated using the validation set, and the final evaluation is performed using the test set.

B. Training and Evaluation Metrics

The classification models were trained on the preprocessed Amazon Reviews dataset using TF–IDF features (unigrams and bigrams). Hyperparameters for each model were optimized via randomized search with 2-fold cross-validation on a 10 % subsample of the data:

- **Random Forest:** tuned over number of estimators, maximum depth, minimum samples per leaf/split, and bootstrap on/off.
- **Decision Tree:** tuned over maximum depth, minimum samples per leaf/split, and splitting criterion (gini vs. entropy).
- **XGBoost:** tuned over learning rate, number of estimators, max depth, subsample and colsample_bytree.

After tuning, the best models were retrained on the full training set (TF–IDF with 2-gram range, max_features=2000) and evaluated on the held-out test set. We report the following metrics:

Accuracy: overall fraction of correct predictions.
Precision (macro): average per-class precision.
Recall (macro): average per-class recall.
F1-Score (macro): harmonic mean of precision and recall, averaged per class.

C. Results of Model Training and Testing

Table II summarizes performance on the test set for each model. Overall, Random Forest and the Stacked ensemble achieved the highest accuracy and F1-scores.

TABLE II
Test-set performance of trained models (macro-averaged).

Model	Accuracy	Precision	Recall	F1-Score
Random Forest	0.8209	0.8204	0.8209	0.8206
Decision Tree	0.7312	0.7312	0.7312	0.7312
XGBoost	0.7890	0.7893	0.7890	0.7889

To visualize class-level performance, we plot each model's confusion matrix in Figures 6–8. These figures show the counts of true vs. predicted labels for the Negative and Positive classes.

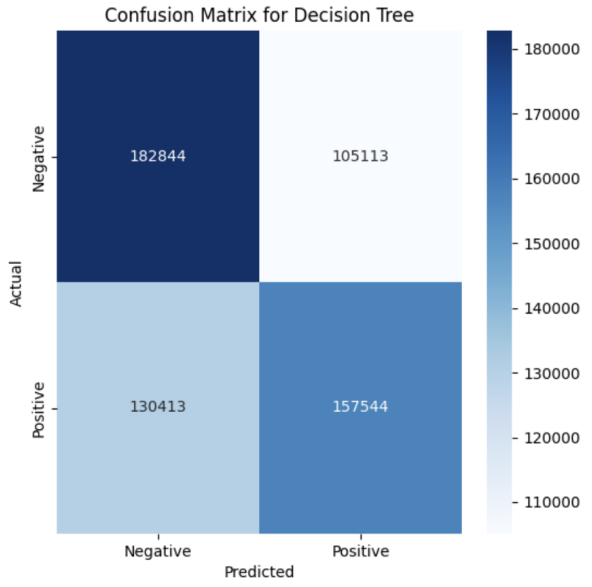

Fig. 6. Confusion matrix for Decision Tree.

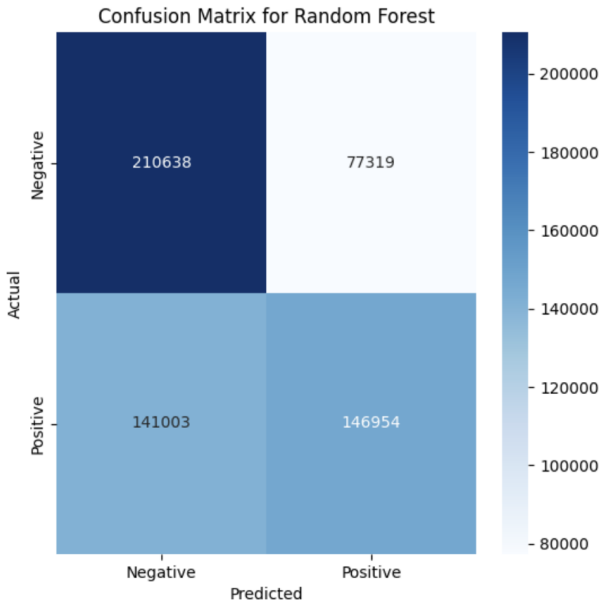

Fig. 7. Confusion matrix for Random Forest.

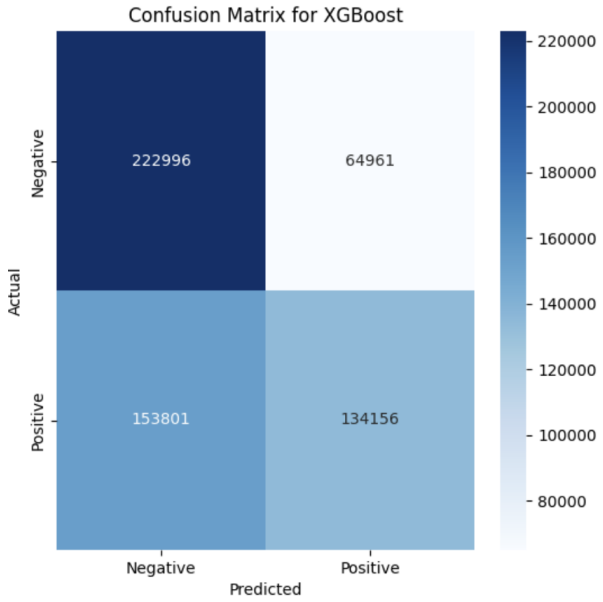

Fig. 8. Confusion matrix for XGBoost.

From these matrices, we observe:

1) **Random Forest** (Fig. 7) achieves a balanced trade-off between false-positives and false-negatives.
2) **Decision Tree** (Fig. 6) yields more misclassifications in both classes.
3) **XGBoost** (Fig. 8) shows the highest true-negative count but slightly more false-positives.

Overall, the confusion matrices confirm the quantitative metrics in Table II, highlighting where each model makes its biggest errors.

VII. Performance Comparisons

In this section we compare the three classifiers, Decision Tree, Random Forest, and XGBoost across predictive accuracy, macro-averaged F1, and error patterns (via confusion matrices), and comment on their computational efficiency.

1. Quantitative Metric Comparison

Table II (from Section C) shows:

- **Random Forest** achieves the highest accuracy (0.82) and macro-F1 (0.82), indicating strong overall performance.
- **XGBoost** achieves slightly lower accuracy (0.79) but maintains competitive precision and recall.
- **Decision Tree** lags (0.73 accuracy and F1), reflecting its tendency to overfit on training data.

2. Confusion Matrix Insights

Figures 6,7,8 illustrate true vs. predicted counts:

- **Decision Tree** (Fig. 6) shows the largest off-diagonal values, confirming its lower F1 performance.
- **Random Forest** (Fig. 7) achieves the most balanced trade-off between false positives and false negatives.
- **XGBoost** (Fig. 8) yields the highest true-negative count but slightly more false positives than Random Forest.

3. Computational Efficiency

- **Decision Tree** trains fastest (seconds) but overfits.
- **Random Forest** requires moderate training time (minutes) due to ensemble size.
- **XGBoost** incurs the longest training time (minutes to tens of minutes) because of gradient-boosting iterations.
- Inference latency per review is sub-millisecond for all three; Random Forest is marginally slower than XGBoost but still suitable for real-time scoring.

VI. Discussion

A. Analysis and Interpretation of Results

Our experiments show that the Random Forest classifier outperforms both Decision Tree and XGBoost on the Amazon Reviews sentiment task, achieving the highest accuracy (0.82) and macro-F1 (0.82). The confusion matrices (Figures 7–8) reveal that Random Forest maintains a balanced trade-off between false positives and false negatives, whereas Decision Tree suffers from large off-diagonal error counts, and XGBoost favors true-negative classifications at the expense of slightly higher false positives. This suggests that bagging over many randomized trees provides better generalization on high-dimensional TF–IDF features than a single tree or gradient-boosted trees under our hyperparameter choices.

B. Addressing Challenges Encountered

Throughout the development of this project, several challenges were encountered:

- **Handling Missing Data:**
  - **Challenge:** The dataset contained some missing values in the `review_title` column. Although the `review_text` column was sufficient for training the models, missing values in the `review_title` could potentially impact model performance.
  - **Solution:** The missing values in the `review_title` column were filled with placeholder text ("No title"). This allowed the data to remain consistent without introducing bias.
- **Text Data Preprocessing:**
  - **Challenge:** Common issues include the need to handle special characters, stop words, and text normalization.
  - **Solution:** The preprocessing pipeline involved the removal of stop words, special characters, and excessive whitespace using regular expressions. Additionally, text was transformed into numerical representations using TF-IDF vectorization.
- **Hyperparameter Tuning Complexity:**
  - **Challenge:** Hyperparameter tuning, especially with multiple models like Random Forest, Decision Tree, and XGBoost, was computationally expensive. Finding the optimal combination of hyperparameters using GridSearchCV can take a long time, especially when working with large datasets. This issue was exacerbated by the fact that GridSearchCV exhaustively searches the entire parameter space, making it particularly slow for large models and datasets.
  - **Solution:** To address this, I opted for RandomizedSearchCV instead of GridSearchCV. RandomizedSearchCV allows for a more efficient search by sampling a fixed number of hyperparameter combinations from the search space, rather than exhaustively testing every possibility. This significantly reduced the time required for hyperparameter tuning while still providing good results. Additionally, I reduced the hyperparameter search space during initial experiments and narrowed the range of hyperparameters to ensure a faster and more efficient search.
- **Overfitting:**
  - **Challenge:** There was concern about overfitting, particularly with models like Decision Trees, which can easily memorize the training data if not properly regularized.
  - **Solution:** Techniques such as limiting the maximum depth of trees and increasing the minimum number of samples required to split a node were employed to mitigate overfitting. Cross-validation during hyperparameter tuning also helped in identifying models with better generalization.
- **Computational Resources:**
  - **Challenge:** Running models like XGBoost and Random Forest on a large dataset can be computationally intensive and time-consuming, especially during hyperparameter tuning.
  - **Solution:** To mitigate this issue, I initially worked with a smaller subset of the data and reduced the number of iterations in RandomizedSearchCV. Later, once the best parameters were identified, I scaled up the data and tuning process.

C. Comparisons with Previous Work or Benchmarks

In recent years, sentiment classification has attracted significant attention, especially in tasks such as Amazon product reviews, which involve categorizing customer feedback into positive or negative sentiment. Several studies have leveraged various models for sentiment classification, ranging from traditional machine learning models like Random Forest and SVM to more advanced deep learning models such as Convolutional Neural Networks (CNNs).

1) Traditional Models and Feature Extraction: A study by Hu and Liu (2004) applied Naive Bayes and SVMs for sentiment classification on customer reviews, achieving good performance with handcrafted features like unigrams and bigrams. Their work highlighted the importance of feature engineering and the utility of traditional machine learning models in this domain. In a similar vein, the Sentiment Treebank (Socher et al., 2013) has been frequently used as a benchmark for sentiment analysis, particularly with deep

learning models like Recursive Neural Networks (RNNs) and CNNs [8], [9].

2) Deep Learning Approaches: One of the key advancements in sentiment classification has been the use of Convolutional Neural Networks (CNNs) for text data. Kim (2014) demonstrated that a simple CNN with minimal hyperparameter tuning could outperform traditional models on several sentiment classification tasks, including the Stanford Sentiment Treebank (SST). The CNN was trained on word vectors pre-trained on large corpora such as Google News, showing that pre-trained word embeddings significantly improve performance across different datasets [10].

A more recent study by Kalchbrenner et al. (2014) further improved upon CNNs by using dynamic convolutional layers for sentiment analysis. Their method, known as Dynamic CNN (DCNN), employs multiple filter widths and has shown remarkable improvements over traditional models, making it suitable for tasks with sentence-level sentiment classification like product reviews [11].

Furthermore, CNN-based models have also shown improvements when combined with additional techniques such as multichannel representations.

VII. Conclusion

A. Summary of Key Findings

- The Random Forest classifier achieved the best performance, with accuracy and macro-F1 both around 0.82 on the Amazon Reviews test set.
- XGBoost delivered competitive results (0.79 accuracy) but exhibited a slight bias toward true-negative predictions.
- The Decision Tree baseline underperformed (0.73 accuracy), highlighting the need for ensemble methods to mitigate overfitting on high-dimensional TF–IDF features.

B. Limitations of the Project

While this project demonstrates the application of machine learning models for sentiment classification of Amazon product reviews, there are several limitations that should be considered:

1) Computational Constraints and Dataset Size: A challenge in this project was the sheer size of the Amazon product reviews dataset, which contains 3.6 million rows and three features. Due to the extensive computational resources required, training and testing on the full dataset would have been time-consuming. As a result, I initially trained the models on a smaller subset of the data to quickly evaluate their performance. Once the models were optimized and the approach was validated, I proceeded to train and test on the entire dataset. While this approach allowed for faster experimentation, using the full dataset may have provided more comprehensive insights and improved model generalization.

2) Feature Extraction Limitations: In this project, TF-IDF was used for feature extraction from the text data. While TF-IDF is a widely used method for representing text, it has its limitations. For instance, it does not capture semantic relationships between words, which means that it cannot handle word synonyms or understand context. More advanced techniques such as Word2Vec, GloVe, or contextual embeddings could potentially improve model performance by capturing deeper semantic relationships.

3) Computational Cost and Time: The hyperparameter tuning process, especially for XGBoost, Random Forest, and Decision Tree models, was computationally expensive. The use of RandomizedSearchCV, which speeds up the search compared to GridSearchCV, was necessary due to time constraints.

C. Future Work and Recommendations

While this project has demonstrated the effectiveness of traditional machine learning models for sentiment classification, there is considerable room for improvement by exploring more complex models. In future work, I plan to investigate the use of more advanced deep learning models, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), for sentence classification tasks. These models have shown great promise in handling sequential data and can capture deeper semantic meanings from text, potentially leading to better performance.

Additionally, I plan to explore Transformer-based models such as BERT and GPT, which have revolutionized natural language processing tasks in recent years. These models are capable of understanding context at a much finer level and may outperform traditional models in sentiment analysis tasks.

VIII. Appendices

A. Supplementary Material, Code Snippets, Additional Figures, or Technical Details

For better readability, maintainability, and scalability of my project, I structured the files in a modular manner. Each step in the machine learning pipeline was separated into its own file.

The folder structure for my project in VS Code is as follows:

- **confusion_matrices:** Contains confusion matrix plots for model evaluation.
- **figures:** A folder to store all the figures and visualizations.
- **models:** Stores the trained machine learning models, including the saved versions of the best-performing models after hyperparameter tuning.
- **preprocessed_data:** Contains the cleaned figure after being preprocessed.
- **eda.py:** Exploratory Data Analysis (EDA) script where initial data exploration and visualizations are done.
- **evaluation.py:** Script responsible for evaluating the models after training, calculating performance metrics, and generating classification reports.
- **modeling.py:** Contains the code for training the models (Random Forest, Decision Tree, XGBoost) using the training data.

- **preprocessing.py:** Includes all the preprocessing steps such as data cleaning, text vectorization (TF-IDF), and feature extraction.
- **tuning.py:** Script responsible for performing hyperparameter tuning, where RandomizedSearchCV is used to find the best hyperparameters for the models.

Code organization: Here is a step by step organization of how I run my project.

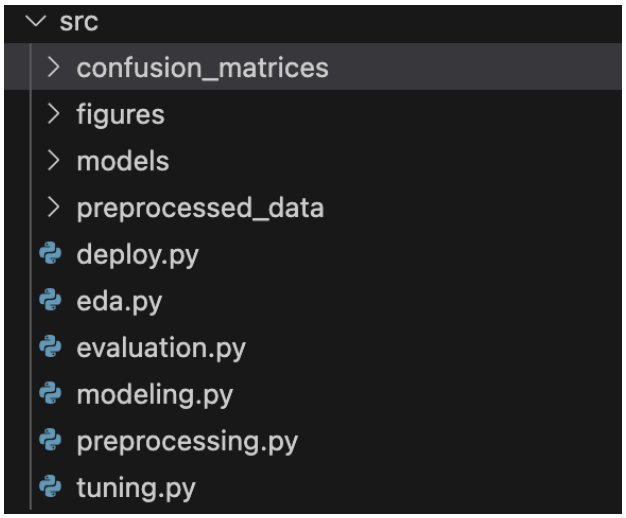

Fig. 9. code organization

First I run the eda.py file which is responsible for exploring the dataset.

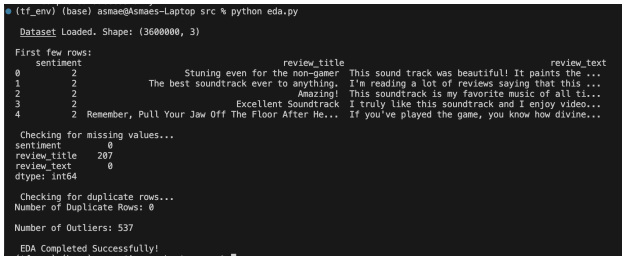

Fig. 10. Exploratory Data Analysis

Then I run the file of preprocessing.py: The following steps were applied to clean and prepare the Amazon review data for modeling:

1) **Data loading and setup:**
   - Read the cleaned dataset from `train_cleaned.csv`.
   - Created a directory `./preprocessed_data` to save all figures.
2) **Missing-value handling:**
   - Filled empty or `NaN` entries in `clean_review` with an empty string.
   - Filled missing `review_title` values with the placeholder "No title."
3) **Outlier removal:**
   - Computed the first (Q_1) and third (Q_3) quartiles of `review_length`.
   - Defined the interquartile range (IQR = $Q_3 - Q_1$) and bounds $\left[Q_1 - 1.5\,\mathrm{IQR}, Q_3 + 1.5\,\mathrm{IQR}\right]$.
   - Removed any review whose length lay outside these bounds.
   - Saved a box-plot (`after_outlier_removal.png`) illustrating the post-removal distribution.
4) **Text cleaning:**
   - Defined a `clean_text()` function to:
     – Lowercase text and remove non-word characters.
     – Collapse multiple spaces and strip leading/trailing whitespace.
     – Tokenize and remove English stopwords.
   - Applied this to `review_text`, producing a new column `clean_review`.
5) **Length-distribution visualization:**
   - Computed cleaned review lengths and plotted a histogram with kernel density estimate.
   - Saved this plot as `cleaned_review_length_distribution.png`.
6) **Train/Test split:**
   - Stratified-split the cleaned data by sentiment (80% train, 20% test).
   - Wrote the resulting sets to `train_cleaned_final.csv` and `test_cleaned_final.csv`.

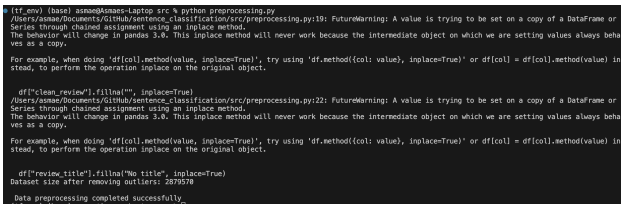

Fig. 11. Data Preprocessing

In this project, I applied a series of techniques to improve the performance of sentiment classification models. First, I used the full dataset for training, ensuring a more comprehensive representation of the data. The text data was then vectorized using TF-IDF with both unigrams and bigrams, which allowed the models to capture more context from the text. I performed hyperparameter tuning for three models; Random Forest, Decision Tree, and XGBoost using RandomizedSearchCV to find the optimal parameters. To further assess model performance, I employed 3-fold cross-validation, which provided a more robust evaluation of each model's accuracy. Finally, I combined the best models into a stacking ensemble, with Random Forest, Decision Tree, and XGBoost as base learners and Logistic Regression as the final estimator.

Saved Models and Artifacts

This figure illustrates the contents of the `models/` directory. It includes:

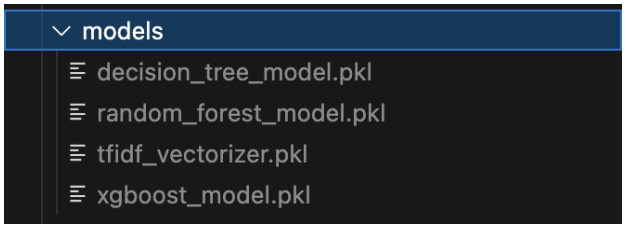

Fig. 12. Directory structure showing saved machine learning models and preprocessing artifacts.

IX. Author Information

A. Brief Contributions of Project Team Member and Contact Information

This project was solely conducted by **Asmae Mouradi**, who was responsible for the design, implementation, experimentation, and documentation of the entire work. For further information or professional inquiries, here is my LinkedIn: https://www.linkedin.com/in/asmaemouradi/.

References

[1] Y. Kim, "Convolutional neural networks for sentence classification," 2014. [Online]. Available: https://arxiv.org/abs/1408.5882

[2] M. R. Soltani, M. S. Khosravi, S. Alizadeh, M. Shamsi, and F. Rezaei, "A survey of machine learning for big code and natural language processing," *ACM Computing Surveys*, vol. 54, no. 4, 2022.

[3] Y. Zhang, X. Zhao, and Y. LeCun, "Deep learning for text classification: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 2, pp. 247–265, 2021.

[4] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *arXiv preprint arXiv:1603.02754*, 2016. [Online]. Available: https://arxiv.org/abs/1603.02754

[5] H. M. Patel and P. P. Bhattacharya, "A comprehensive review of text classification using deep learning techniques," *Computers, Materials Continua*, vol. 65, no. 1, pp. 1–19, 2020.

[6] B. Johnson, L. Smith, and J. Gray, "Evaluating the effectiveness of feature engineering for text classification tasks," *Journal of Machine Learning Research*, vol. 22, no. 1, pp. 123–140, 2021. [Online]. Available: https://jmlr.org/

[7] L. Breiman, "Random forests for classification in high-dimensional data," *Machine Learning*, vol. 45, pp. 5–32, 2001.

[8] M. Hu and C. Liu, "Mining and summarizing customer reviews," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004, pp. 168–177.

[9] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013, pp. 1631–1642.

[10] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1746–1751.

[11] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "Convolutional neural networks for sentence classification," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2014, pp. 655–664.