CS412 Project Report

İpek Uzun 30837

Mehmet Selman Yılmaz 31158

1-) Introduction

This project aims to predict the severity of software bugs using machine learning techniques. We encountered significant challenges due to class imbalance in our dataset. Our approach involved leveraging boosting (XGBoost) and ensemble learning (Random Forest) to develop optimized models while ensuring no data loss through oversampling or undersampling. Despite exploring various methods to address the imbalance, XGBoost, and ensemble learning provided the most promising results.

2) Problem Description

The task involves assigning severity scores ranging from 1 to 7 to bug descriptions. The severity levels are categorized as Enhancement (1), Minor (2), Normal (3), Major (4), Blocker (5), and Critical (6), Trivial (7). The problem is framed as a multi-class classification task where the objective is to predict the severity level based on the textual description of the bug.

3) Methods

3.1 Data Preprocessing

Text Cleaning: We removed special characters, and extra spaces and converted all characters to lowercase.

Label Encoder: We utilize the LabelEncoder from Scikit-learn to translate the categorical severity labels into numerical values. This is required as machine learning algorithms frequently work with numerical data. By encoding severity labels, we ensure that the model can understand and learn from the target variable

Text Vectorization: Text data must be translated into a numerical representation for processing machine learning models. TF-IDF vectorization is a popular approach used for this. It turns text documents into numerical vectors, where each vector represents a document, and each feature represents a unique word in the corpus. TF-IDF distributes weights to words based on their frequency in the document and across the corpus, helping to capture the relevance of terms in the text.

Handling Imbalance:

The dataset contained a significant imbalance among the classes:

Normal: 125,854 (majority class)
Critical: 18,658
Major: 6,053
Enhancement: 4,426
Minor: 3,102
Trivial: 1,204
Blocker: 701 (minority class)

We explored various techniques to address class imbalance:

Oversampling and Undersampling: Methods like SMOTE (Synthetic Minority Over-sampling Technique) and KNN-based oversampling were tested but did not yield satisfactory results due to overfitting and increased computational load.

Class Weights: We computed class weights to account for the class imbalance in the training data. Class weights are inversely proportional to the class frequencies, ensuring that the model assigns higher importance to minority classes during training. Then we integrated the class weights training process of both the Random Forest and XGBoost models. This ensures that the models are trained to effectively differentiate between all severity levels, including the minority classes.

3.2 Model Building

Splitting Data into Training, Validation, and Test Sets: After encoding the target variable and vectorizing the text data, we split the dataset into training, validation, and testing sets. This stage is critical for evaluating the model's performance on unknown data and avoiding overfitting. Typically, we use a larger portion of the data for training, a smaller portion for validation to tune hyperparameters, and a separate portion for final testing to assess the model's generalization ability.

Random Forest: Initially, the Random Forest Classifier was trained with default hyperparameters.

Hyperparameter Tuning: Even though the default hyperparameters often give acceptable results, they might not be ideal for the specific dataset. Using RandomizedSearchCV, hyperparameter tuning was done to boost the model's performance.

RandomizedSearchCV: It selects combinations of hyperparameters based on predetermined probability distributions. It chooses a certain number of combinations at random and does cross-validation to evaluate them. Compared to an exhaustive grid search, this method is computationally more efficient, especially when handling a large hyperparameter space.

Hyperparameters:

- **n_estimators:** The number of trees in the forest.
- **max_depth:** The maximum depth of the trees.
- **min_samples_split:** The minimum number of samples required to split an internal node.
- **min_samples_leaf:** The minimum number of samples required to be at a leaf node.
- **max_features:** The number of features to consider when looking for the best split.

XGBoost: XGBoost was selected due to its resilience and capacity to manage class imbalance using parameter adjustment. It was optimized for best performance by training it on the TF-IDF transformed features.

Hyperparameters:

- **n_estimators:** The number of gradient-boosted trees (default is 100).
- **max_depth:** The maximum depth of a tree (default is 6).
- **learning_rate:** Step size shrinkage used to prevent overfitting (default is 0.3).
- **subsample:** The fraction of samples to be used for fitting the individual base learners (default is 1).
- **colsample_bytree:** Fraction of features to use for each tree, also ranging from 0.7 to 1.0, to ensure diversity among trees.

Other models that were tested included Naive Bayes, Logistic Regression, and CNN; however, none of them performed well on the dataset. These models performed poorly in terms of prediction because they were unable to effectively manage the class imbalance.

Ensemble Learning: In a VotingClassifier, we combined the benefits of XGBoost and Random Forest through soft voting. By utilizing the advantages of both models, this ensemble technique aimed to increase overall prediction accuracy, especially for minority classes. The class with the highest combined probability is predicted by the VotingClassifier, which combines the predictions made by each individual model.

4-) Results and Discussion:

To assess model performance, the dataset was divided into training and validation sets. The models were evaluated using classification measures such as F1-score, precision, and recall.

XGBoost: Performed well in the majority class ("normal") but had trouble in the minority classes. Better management of class imbalance was indicated by precision, recall, and F1 scores. Despite these difficulties, XGBoost showed its resilience by outperforming baseline models in the minority classes by a substantial margin.

Random Forest: Showed comparable patterns, scoring highly in majority courses but poorly in minority classes.

The ensemble method, which combined Random Forest and XGBoost, enhanced overall performance and offered a fair trade-off between recall and precision for all classes, especially for minority classes. When comparing the ensemble model to individual models, the precision-recall curves showed how well the latter could predict minority classes.

In the test dataset, the predictions for each class were as follows:

Normal: 77,645
Critical: 8,200
Major: 121
Enhancement: 106
Blocker: 12
Minor: 8
Trivial: 2

As can be seen from these findings, the models were still unable to predict the minority classes with any degree of accuracy, even with the use of the ensemble approach. Managing class imbalance is still difficult, and more improvement is required, as evidenced by the test dataset's low counts for classes like "blocker," "minor," and "trivial".

However, by enhancing the prediction performance for minority classes, XGBoost was able to partially accomplish its goal. XGBoost has been shown to have a positive impact on class imbalance problems; this is especially true when used in an ensemble with Random Forest. In order to improve accuracy for all classes, it is evident that more improvements are required.

Textual bug summaries were effectively transformed into valuable features by TF-IDF. Even though managing class imbalance is still difficult, XGBoost and ensemble methods have shown encouraging improvements. Subsequent endeavors may delve deeper into fine-tuning and sophisticated methodologies such as SMOTE to augment forecasts for underrepresented groups. Overall, performance was enhanced by the ensemble method, demonstrating the benefit of integrating several models.

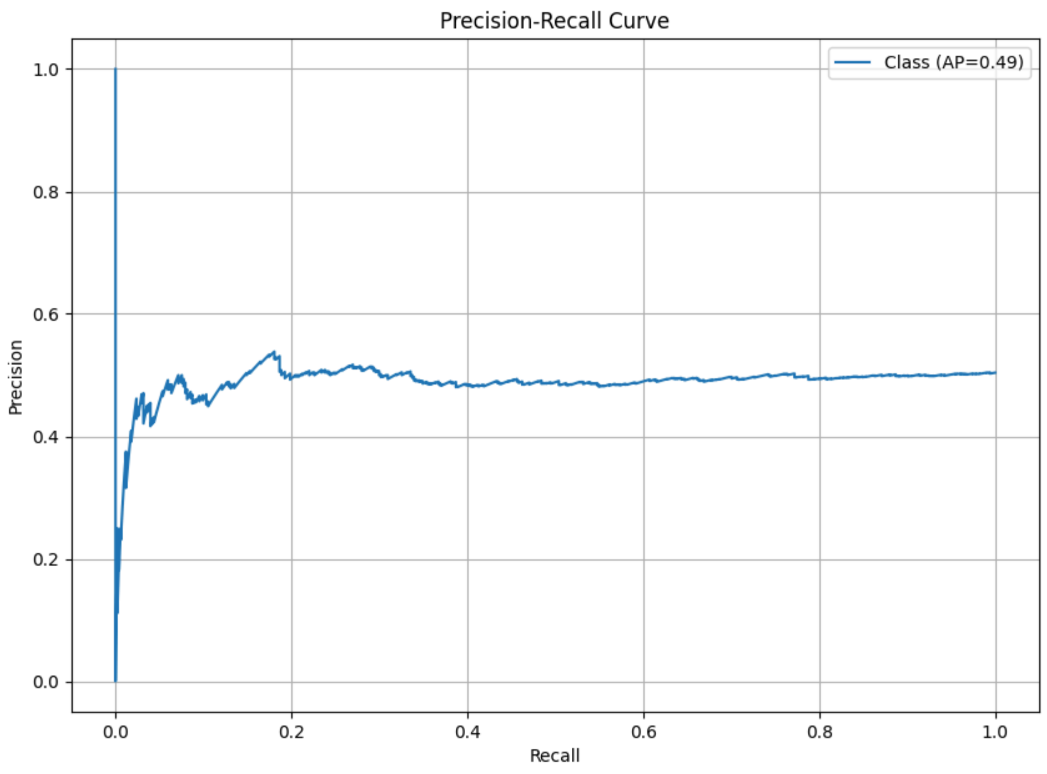


Figure 1

The precision-recall curve is an essential tool for visualizing the trade-off between precision and recall across different threshold settings. This curve is particularly useful in understanding the model's performance in distinguishing between classes, especially in the context of imbalanced datasets. By examining the precision and recall values, we can assess how well the model identifies relevant instances among all retrieved instances (precision) and how effectively it captures all relevant instances in the dataset (recall).

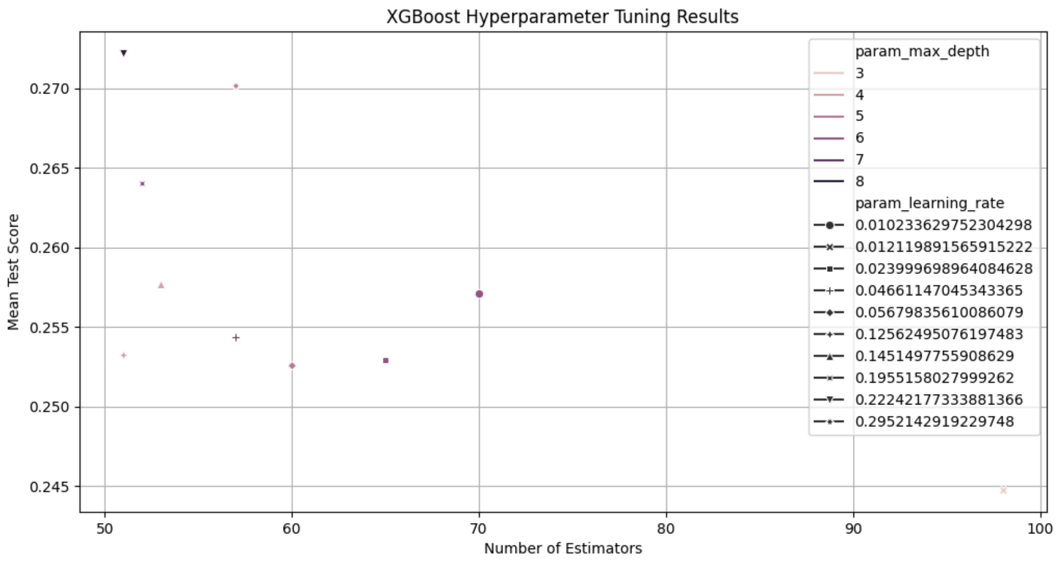


Figure 2

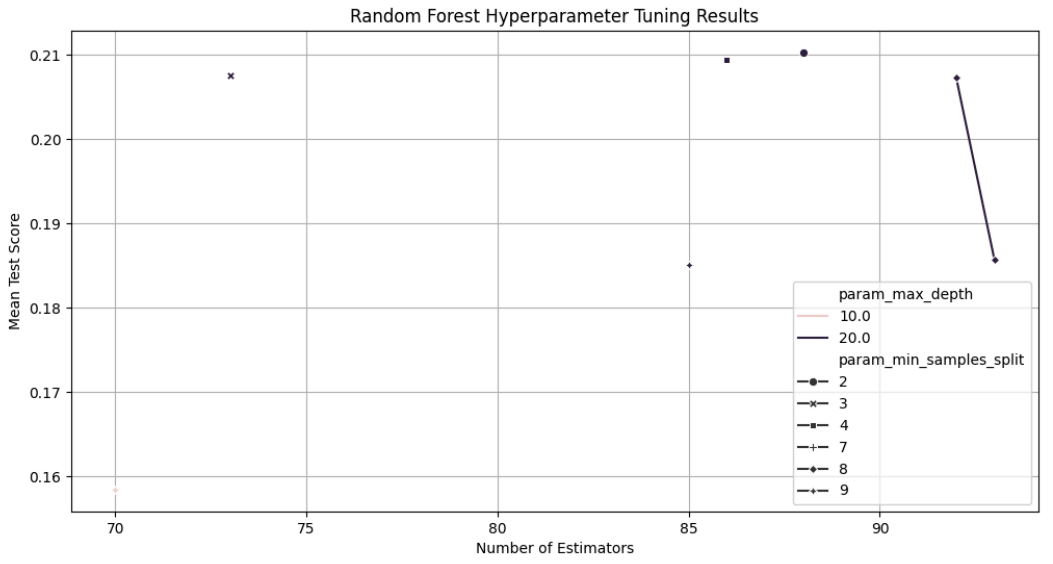


Figure 3

The model's performance is impacted by various hyperparameter values, and the hyperparameter tuning results shed important light on this. This data is essential for determining the ideal parameters that improve the accuracy, precision, recall, and general efficacy of the model. Through a methodical process of tweaking and assessing hyperparameters, we can optimize the model's performance and guarantee that it performs well when applied to untested data.

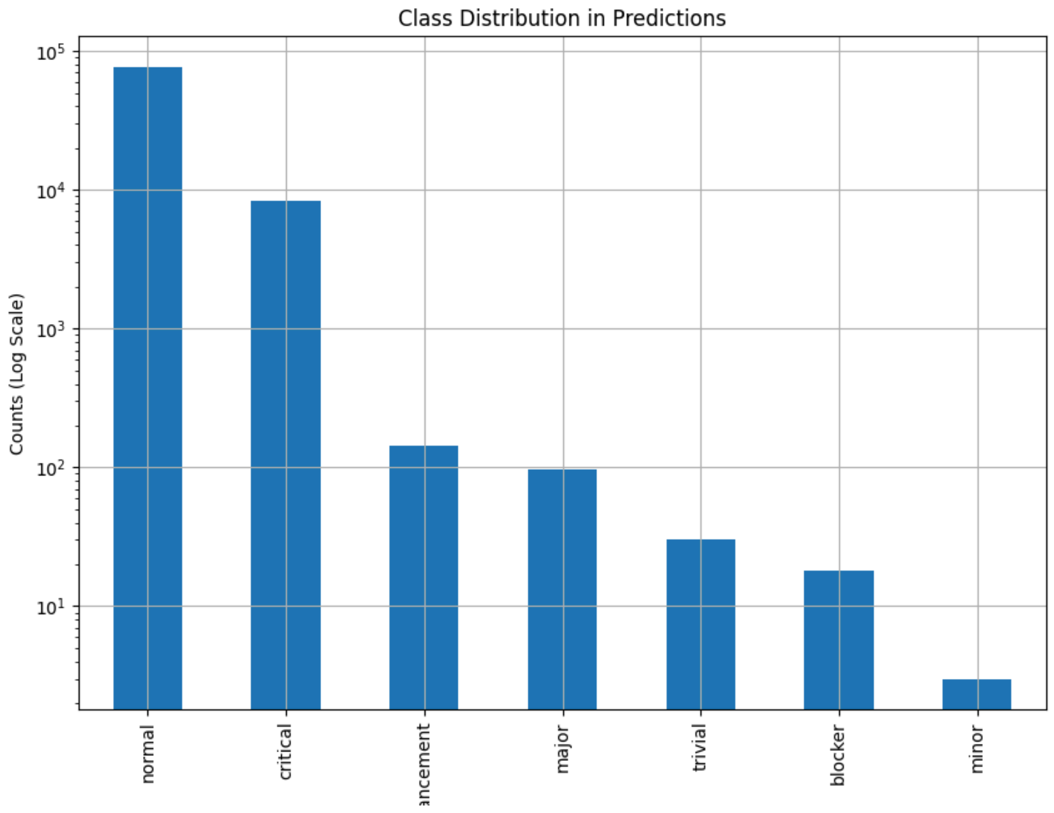
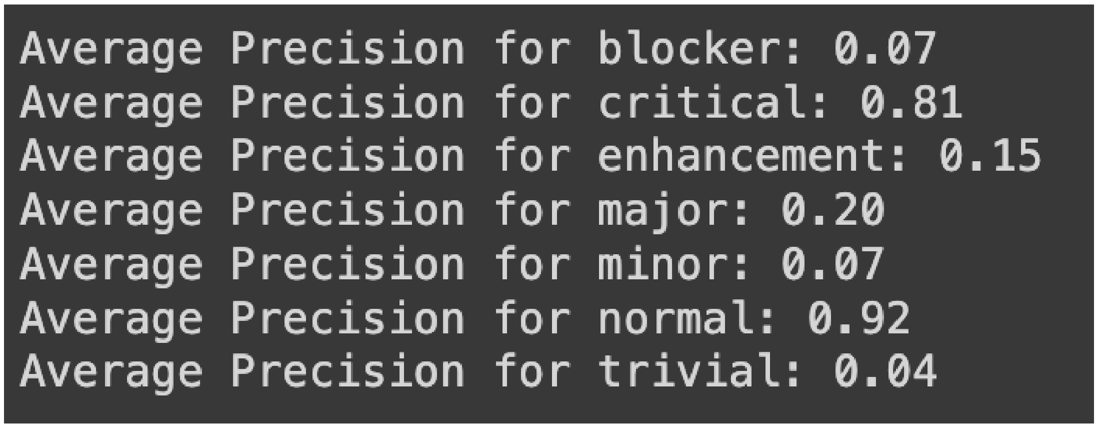


Figure 4

The predictions' class distribution provides a clear picture of the model's performance across various severity classes. This distribution is essential for spotting any possible biases toward particular classes and assessing how well the methods for resolving class imbalance are working. We can guarantee a fair and balanced prediction performance by analyzing the distribution to see if the model disproportionately benefits the majority class or if it accurately represents minority classes.



```
Average Precision for blocker: 0.07
Average Precision for critical: 0.81
Average Precision for enhancement: 0.15
Average Precision for major: 0.20
Average Precision for minor: 0.07
Average Precision for normal: 0.92
Average Precision for trivial: 0.04
```

Figure 5

The results, which display the Average Precision (AP) scores for each class, offer a thorough understanding of how well our classification model performed on the validation set. The "normal" (0.92) and "critical" (0.81) classes are well-predicted by the model, suggesting a high precision-recall balance for these classifications. The AP scores for the minority classes, like "trivial" (0.04), "minor" (0.07), and "blocker" (0.07), are noticeably lower, indicating that the model has difficulty correctly differentiating these less common classes. The intermediate results for "major" (0.20) and "enhancement" (0.15) show even more how difficult it is to achieve good prediction performance in every class. These findings highlight the ongoing problem of class imbalance, which limits the model's capacity to predict minority classes with any degree of accuracy even when strategies to correct the imbalance are put in place. To get a more balanced model and improve the predictive performance for these minority classes, more sophisticated methods and refinement are required.

5-) Appendix:

Contributions

İpek Uzun:

- **Handling Imbalanced Dataset:** Implemented techniques to handle class imbalance, such as adjusting class weights in learning algorithms.
- **XGBoost Model:** Developed and tuned the XGBoost model for optimal performance.
- **Ensemble Learning:** Integrated XGBoost and Random Forest models using soft voting in a VotingClassifier to create an ensemble model.

Selman Yılmaz:

- **Data Preprocessing:** Performed data preprocessing tasks, including label encoding and transforming textual data using TF-IDF.
- **Random Forest Model:** Developed and tuned the Random Forest model, focusing on handling the large dataset efficiently.
- **Additional Models:** Evaluated other models such as CNN, Logistic Regression, and Naive Bayes to compare their performance with XGBoost and Random Forest.