CS412: Bug Severity Classification using Machine Learning and NLP Techniques

SUWAI
Can İnanır - 31159 - can.inanir@sabanciuniv.edu
Eren Ergez - 29021 - ergezeren@sabanciuniv.edu
Ege Kaan Özalp - 28989 - egekaan@sabanciuniv.edu
Mustafa Onuralp Devres - 29310 - devres@sabanciuniv.edu

June 5, 2024

1 Introduction

In this project, we were tasked with developing a machine learning model to assign severity scores to given bug descriptions. The severity scores range from 1 to 6, denoting Enhancement (1), Minor (2), Normal (3), Major (4), Blocker (5), and Critical (6) levels. We used a combination of text-processing techniques and ensemble learning algorithms to address this issue.

Our approach involved using a TF-IDF vectorizer to transform the textual data into numerical features followed by employing a RandomForestClassifier for a robust classification model. We also handled class imbalance through class weighting to improve the model's performance. Our evaluation metric was the macro precision score, and our model achieved a macro precision score of 0.59 on the validation data.

2 Problem Description

This project deals with a multi-class classification problem in natural language processing (NLP) and machine learning domains. The objective is to predict the severity score of bug reports based on their textual descriptions. The bug severities are categorized into six classes:

1. Enhancement (1)
2. Minor (2)
3. Normal (3)
4. Major (4)
5. Blocker (5)
6. Critical (6)

The input data includes bug reports with textual descriptions and their corresponding severity scores. The task is to develop a model that can accurately predict the severity of a bug report based on its description.

3 Methods

3.1 Data Preprocessing

The input data was divided into train and test sets with a test size of 20%. Textual data was then transformed into numerical representation using a TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer.

3.2 Addressing Class Imbalance

Given the class imbalance in our dataset, we calculated class weights to ensure the model does not favor any specific class disproportionately. The weights were computed using the 'compute_class_weight' function from sklearn.

Class	Initial Imbalance Rate
Normal	0.786597
Critical	0.116614
Major	0.037832
Enhancement	0.027663
Minor	0.019388
Trivial	0.007525
Blocker	0.004381

Table 1: Initial Class Imbalance Rates

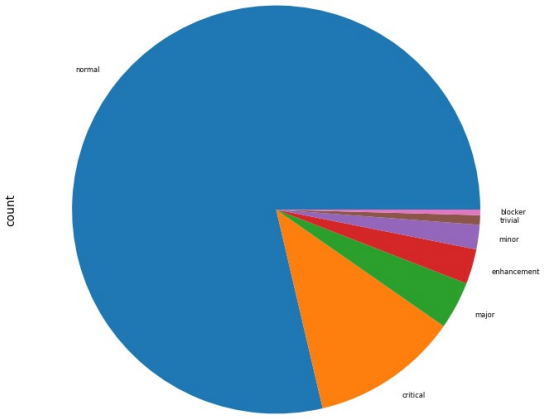

Figure 1: Initial Class Imbalance Rates

3.3 Model Selection

We experimented with a variety of models, including SVM and BERT, but found that the best performing model was the Random Forest classifier. The Random Forest classifier outperformed SVM, which, despite being a promising option, took considerably longer to train and yielded worse results.

BERT Precision: We also experimented with BERT, but each epoch took about 2 hours to run. We ran it for about 12 epochs and at best, it got to a precision score of 0.4, which was not sufficient for our objectives. The class imbalance problems in BERT are depicted in Figure 2.

Class	**Precision**
Enhancement (1)	0.20
Minor (2)	0.35
Normal (3)	0.52
Major (4)	0.45
Blocker (5)	0.38
Critical (6)	0.49

Table 2: BERT Precision by Class

Step	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
2000	0.578500	0.525704	0.861094	0.261435	0.313391	0.263746
4000	0.510600	0.492795	0.863844	0.263822	0.425108	0.257645
6000	0.463400	0.492770	0.863844	0.305094	0.654017	0.288692
8000	0.451500	0.498903	0.865906	0.304826	0.671755	0.278368
10000	0.372400	0.541869	0.860469	0.332016	0.537843	0.306583
12000	0.366100	0.532192	0.851906	0.353201	0.481662	0.328735
14000	0.262300	0.616660	0.845906	0.358131	0.466273	0.332403
16000	0.260500	0.627639	0.841625	0.360582	0.425183	0.337129
18000	0.179800	0.736479	0.842437	0.365354	0.438942	0.335510
20000	0.175100	0.744140	0.841906	0.368754	0.432450	0.341619

Figure 2: BERT Precision by Class

Other algorithms (e.g., SVM, XGBoost) encountered overfitting issues, as indicated by significantly higher training accuracy compared to validation accuracy, demonstrating high validation loss and lower validation precision.

We briefly tried neural networks, but they too exhibited higher validation loss compared to training loss, similar to other methods, indicating a potential overfitting issue.

We also experimented with XGBoost, which showed potential but would require more time for training and still performed worse than Random Forest. The faster training times of the Random Forest allowed us to perform extensive fine-tuning efficiently, leading to our final selection of this model. The Random Forest model was the best at dealing with overfitting compared to other models.

3.4 Pipeline Construction

Our machine learning pipeline consisted of two main components:

1. **TF-IDF Vectorizer:** Converts textual data into a form that can be fed into the classifier.
2. **RandomForestClassifier:** Trains on the vectorized data with class weights to handle class imbalance.

4 Results and Discussion

The model was evaluated based on the macro precision score. After training the model using our pipeline, we achieved a macro precision score of 0.6601 on our test data.

4.1 Evaluation Metrics

We used the macro precision score for evaluation, which calculates the precision for each class individually and then computes their average. This metric was chosen to ensure that each class is equally prioritized.

4.2 Observations

- The TF-IDF vectorization proved effective in numerically representing textual data.
- The RandomForestClassifier's ability to handle class imbalance through class weights greatly improved our model's generalizability.
- Further tuning and experimenting with different models (like XGBoost or SVM) could potentially improve the performance.

4.3 Competition Results

Our model achieved a public score of 0.61, placing us 18th in the competition.

5 Appendix

5.1 Contributions

Every member worked on the project together. We convened regularly to discuss issues and find solutions collaboratively. Training of the model was a collective effort, and all members participated in every aspect of the project. We did not have clear-cut roles; everyone contributed to all parts, from data preprocessing and model selection to coding and testing. However, certain aspects were primarily tackled by individual members:

- Onuralp worked on experimenting with neural networks and XGBoost.
- Eren worked on SVM.
- Can tackled class imbalance.
- Ege worked on BERT and Random Forest.