CS412 Term Project Report

İdil Kara - 30803
Sima Adleyba - 28889
İdil Esin Aydın - 30647
İdil Yelçe - 28845

Introduction

The task for the project was to assign the severity level to a given bug description. There were 6 levels of severity, which are Trivial (0), Enhancement (1), Minor (2), Normal (3), Major (4), Blocker (5), and Critical (6), in order. We were to use different feature processing and classification methods and the results were to be evaluated based on macro precision.

Problem Description

As mentioned above, the problem was to assign the severity level to a given bug description. As we were to assign a level of severity, which is a discrete value from 0 to 6, we considered this as a classification problem. However, we did not assign encodings as ordinal values with our approach.

Methods

Preprocessing

We first processed the characters of the bug description in the file as the original version contained stopwords, punctuations and non-ASCII characters. We also applied stemming to the words. Then we tried both “tokenizer” from the “tensorflow.keras.preprocessing.text” library and TfidfVectorizer to tokenize the words to turn them into features. In our best model, we used the latter.

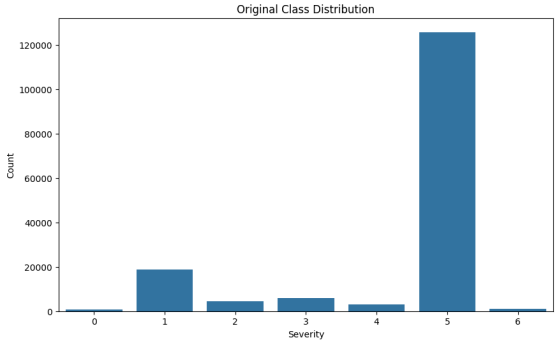
Then, we checked the data and found that there was a huge class imbalance, as can be seen from the image on the right. To handle this, we used RandomOverSampler from the imblearn library, which calculates the counts of each class in the training labels using counter and defines a target count for specific classes to ensure that each specified class will have a certain number of samples after resampling, resulting in a new, balanced dataset.

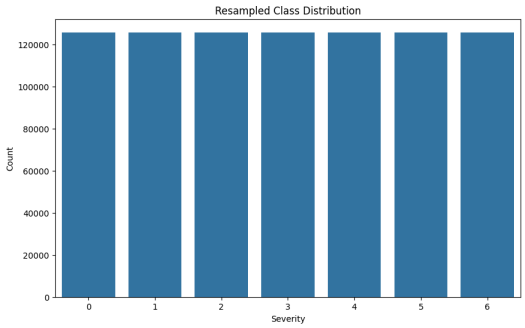
```
Distribution of labels in y_train:
severity_encoded
5    100534
1     14995
3      4852
2      3574
4      2509
6       976
0       558
Name: count, dtype: int64
```

Class Imbalance

Handling Class Imbalance in Train

For handling class imbalance we tried various methods that weren't all successful. We tried approaches that modified the train set such as SMOTE, oversampling via RandomOverSampler. Here's the class distribution after SMOTEto display how it handles class imbalance:





However, we noticed that it was taking so much time to train the models using oversampling methods. With undersampling it would throw away the %80 of the training data so we didn't prefer it. In our final models, we used class weights to handle class imbalance.

Hyperparameter Tuning

We tried grid search with most of the models we tried, such as random forest, logistic regression or xgboost. Below is an example of how we did hyperparameter tuning with xgboost.

For XGBoost
param_grid = {
'colsample_bytree': [0.6, 0.8, 1.0],
'learning_rate': [0.01, 0.05, 0.1],
'max_depth': [3, 5, 7],
'n_estimators': [500, 750, 1000],
'subsample': [0.6, 0.8, 1.0]
}

For the hyperparameter tuning we did a grid search with cross validation with the values above and saw that

params = { 'colsample_bytree': 0.8, 'learning_rate': 0.01, 'max_depth': 5,
'n_estimators': 750, 'subsample': 0.8 }
generally give the best results.

We realized grid-search was very time-consuming and inefficient for the large dataset so we continued with different methods after.

Model Description

We tried different combinations of some classification models such as xgboost, catboost, light gbm classifier, logistic regression, and random forest and experimented with different transformers such as distillBERT and fasttext's transformers where we also tried using building neural network models.

After trying different approaches and failing, we tried to bring together some of our best models. We first built XGBoost with and without weights and achieved %70 macro precision as shown below.

	precision	recall	f1-score	support
0	0.33	0.02	0.04	143
1	0.77	0.73	0.75	3663
2	0.77	0.01	0.02	852
3	0.85	0.01	0.03	1201
4	1.00	0.00	0.01	593
5	0.87	0.97	0.92	25320
6	0.35	0.04	0.06	228
accuracy			0.85	32000
macro avg	0.70	0.25	0.26	32000
weighted avg	0.85	0.85	0.81	32000

XGBoost model without class weights's classification scores

Macro Precision Score: 0.7153

	precision	recall	f1-score	support
0	0.50	0.01	0.01	143
1	0.77	0.76	0.77	3663
2	1.00	0.01	0.01	852
3	0.87	0.01	0.02	1201
4	0.00	0.00	0.00	593
5	0.87	0.97	0.92	25320
6	1.00	0.00	0.01	228
accuracy			0.86	32000
macro avg	0.72	0.25	0.25	32000
weighted avg	0.84	0.86	0.82	32000

XGBoost with class weights model's classification scores

We concluded that it was enough for a weak classifier in our model. Then, we also tried logistics regression with different weighting to see if we can improve precision for different classes. Both gave around %70 macro precision results.

Here is the code sample for the class weights we implemented for logistic regression.

```
#lr1:
class_weight_lr1 = {0: 3, 1: 5, 2:2, 3:1, 4:1, 5:4, 6:5}
log_reg_model_fin1 = LogisticRegression(class_weight=class_weight_lr1, random_state=42, max_iter= 70)
log_reg_model_fin1.fit(X_tfidf_train_fin,labels_fin)

#lr2:
class_weight_lr2 = {0: 4, 1: 5, 2:2, 3:1, 4:1, 5:1, 6:8}
class_weight_lr2[5] = 4
log_reg_model_fin2= LogisticRegression(class_weight=class_weight_lr2, random_state=42, max_iter= 200)
log_reg_model_fin2.fit(X_tfidf_train_fin,labels_fin)
```

After some discussion, we thought it would be good to combine these 2 different weighted logistic regression models with the 2 (weighted and non-weighted) XGBoost models. We trained a CatBoost model by using these four models' predictions then we achieved %91 macro precision on our validation data. Then on kaggle, it actually gave the best private result we had. We wanted to try cross validation but it was both computationally expensive and time-consuming and after trying some parameters we concluded %91 for the macro precision is the best we got. Classification scores for the ensemble model with CatBoost model as the meta learner.

While exploring different gradient boost models, we found the LightGradientBoost classifier was fast and had high accuracy. When we tried it after tuning, it gave our best score on the public test data. However, the score got lower than our best model we described, as the private set is included in the end.

Results and Discussion

Handling Class Imbalance in Evaluation Metrics

Another thing we kept in mind due to the imbalance was the evaluation metrics used. Due to the imbalance most “normal” type bugs were rightfully classified giving quite high but deceptive accuracy therefore we calculated better alternatives for imbalanced datasets such as f1 score, recall, and lastly macro-precision which was our base metric.

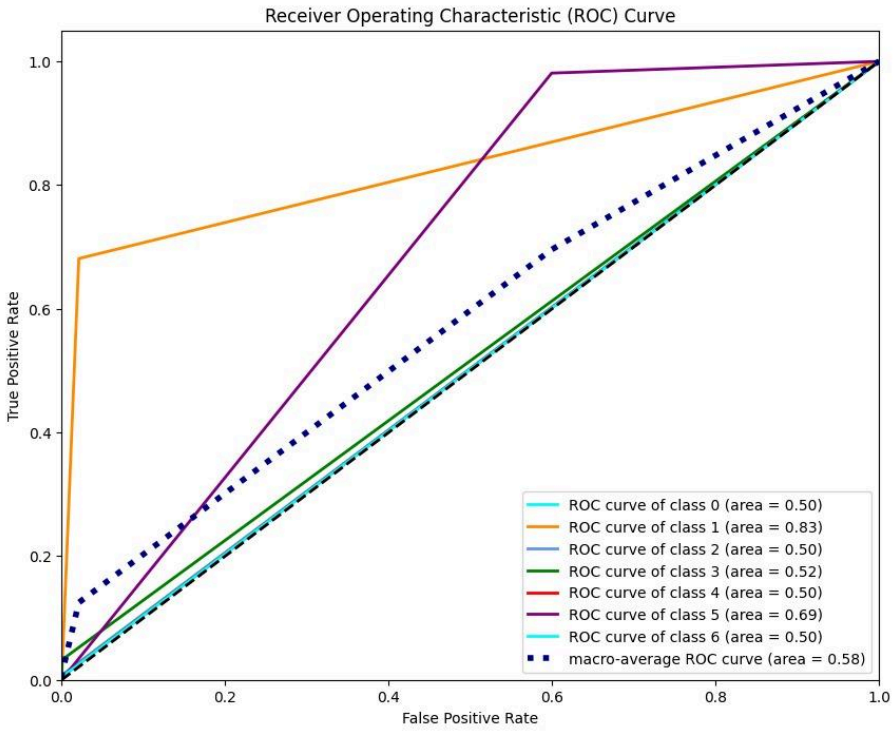
Model Results

After determining that our main focus was on the ensemble method with XGBoost and logistic regression, and the LightGradientBoost model, we tried to improve them with trying different parameters. Our best result was with LightGradBoost, we got 0.91 precision as shown below and our best public score was with this method with 0.60768 macro precision.

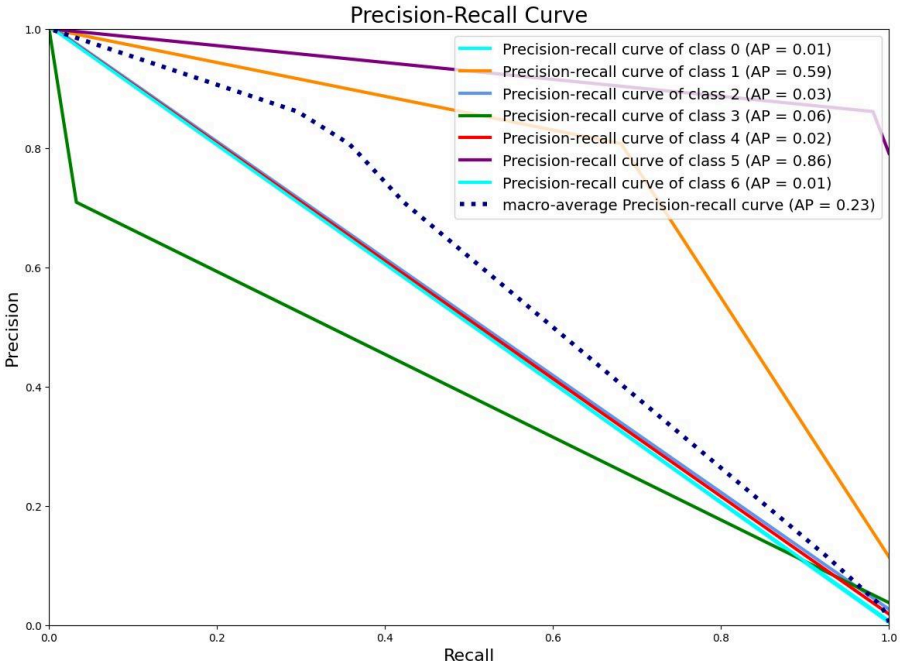
	precision	recall	f1-score	support
0	1.00	0.01	0.01	143
1	0.81	0.68	0.74	3663
2	1.00	0.01	0.01	852
3	0.71	0.03	0.06	1201
4	1.00	0.01	0.01	593
5	0.86	0.98	0.92	25320
6	1.00	0.00	0.01	228
accuracy			0.86	32000
macro avg	0.91	0.25	0.25	32000
weighted avg	0.86	0.86	0.81	32000

Macro Precision Score: 0.9111

However, we suspected that there was an overfit to the validation set with this result. The ensemble method, which we used CatBoost for, did not give a better result than the LightGradBoost in the public macro precision, as it gave 0.51789 macro precision on the public scores. Here is the ROC curve we got from the CatBoost ensemble method.



And here is the Precision-Recall curve we obtained from the CatBoost ensemble classifier



Conclusion

In the end, we submitted our predictions with the tuned LightGradientBoost as it gave the best results in the public test scores, however, our ensemble model with CatBoost performed the best in the private testing (67.08 % on Kaggle). We concluded that it is important to have trust in our model and not solely trust the public test score as it can give deceptive results.

Appendix

The class labels encoding correspondence:

```
{'blocker': 0, 'critical': 1, 'enhancement': 2, 'major': 3, 'minor': 4, 'normal': 5, 'trivial': 6}
```

Work Distribution:

To be able to identify what works and what does not, we decided to distribute the models. After experimenting, we discussed the results we got from varying methods.

Throughout this process, we notified each other and discussed what we could do to improve our models and settled on one approach at the end.

Sima Adleyba

I focused on doing ensemble models with more basic models such as xgboost, bayes, logistic regression and random forest.

For the preprocessing part, I cleaned the non-ASCII characters and stopwords. Then I used tokenizer" from the "tensorflow.keras.preprocessing.text" library to tokenize words and use them as features. To handle class imbalance, I used SMOTE.

After that, I worked on Logistic Regression, Gaussian Bayes, xgboost and random forest alone. I also tried 5 fold cross validation for logistic regression and xgboost, however it was not as effective as I expected and very time consuming.

Lastly, I experimented with different combinations on these and settled with xgboost, random forest and gaussian bayes with hard voting. However, though results on training data was promising, it was not good on the test data.

İdil Esin Aydın

I mainly focused on applying neural networks to the classification problem. I tried different oversampling methods with the nn's like oversampling and SMOTE but they didn't give good results with nn. I also experimented with fasttext tokenizer to see if we would get different results since especially the ensemble models we were trying to run were having very long train periods. Lastly experimented with feature extraction with word count and feature length with the submitted model but it wasn't successful.

İdil Kara

I first tried to build a neural network model using distillBert's transformer however, due to technical difficulties using Cuda I had to stop trying to optimize it. Then I worked on building ensemble models. I tried building stacking models with Naive Bayes classifier and logistic regression models which were good to start with. Later, I tried to train support vector machine models but I did not get any good results with the validation set.

After carefully studying the confusion matrices, I tried different methods to resolve the class imbalance problem. I tried to undersample some classes and oversample some minority classes based on the knowledge on the internet however, it was not very helpful and practical. Then I worked more on the models that utilized class weights that are gradient boost and logistic regression models.

Finally, I switched my focus on gradient boosting methods and building ensemble methods using them. I trained and tuned xgboost, lightgbm and catboost models. In the end, I organized the code that explains our final model, the ensemble model that uses catboost model as meta learner and xgboost and logistic regression models as the weak learners.

I also worked on the details we could do in the preprocessing step, what might be good to remove from the text, and what may be important. I decided to use tfidf vectors for my experiments.

İdil Yelçe

I first tried some simple Logistic Regression model where I did not get a significant result. Then I applied hyperparameter tuning and got .75 precision, but when I applied the model on the provided test data I got a really low score. Then I tried to implement SVM model with hyperparameter tuning, but the complexity of the model was so high that the machine that I was running on could not handle it. After that I tried some feature extraction methods and played with the training data, however did not go much far with my initial logistic regression model.

Lastly, I tried an ensemble model using decision tree and random forest as base models. This approach gave a more significant result compared to my other ones. However, again, when I tried hyperparameter tuning the training took nearly 24 hours and ı did not go any further.