Introduction: A quick summary of the problem, methods, and results.

Problem description: The description of the problem. Formally define how you cast the problem, classification, regression, etc.

Methods: Description of methods, you may use subsections.

Results and Discussion: The results of “ applying the methods to the data set. Include the questions your experiments are designed to answer, the details of the experiments, and your observations. For example, you will be evaluated on AP, including the precision-recall curves. For example, report your findings if you tried different feature processing techniques. Interpretation and discussion of the results.

Appendix: A clear description of the contribution of each person. You may also include extra material if you have many plots and tables.

The task is to assign a severity score to a given bug description. Each bug report includes a description and its bug type. Bug severity should be scored from 1 to 6, from Enhancement (1), Minor (2), Normal (3), Major (4), Blocker (5), and Critical (6), in order.

Example of the data that was worked on

**Introduction:**

In this project, we aimed to develop a machine learning model to predict the severity of bug reports, classified from Enhancement to Critical. The model prediction was achieved by training with the data sets bug_train and testing on the bug_test.

To achieve this, we employed a Random Forest classifier and used the optimized Random Forest as the base estimator for Adaboost.

Initially, our model achieved a macro precision score of 0.69 on the Kaggle competition leaderboard. However, subsequent evaluations resulted in a reduced score of 0.54

**Problem description**

The primary objective is to accurately assign a severity score to a given bug description. Each bug report contains a summary and a type, which need to be analyzed to predict the severity. T bug severity should be scored from 1 to 6, from Enhancement (1), Minor (2), Normal (3), Major (4), Blocker (5), and Critical (6), in order
A fragment of the data that was worked upon is shown below:

| bug_id | summary | severity |
|---|---|---|
| 365569 | Remove workaround from bug 297227 | normal |
| 365578 | Print Preview crashes on any URL in gtk2 builds | critical |
| 365582 | Lines are not showing in table | major |

The training and predictions were performed by using data in this format where each column represented bug_id, summary and severity, in order. At last during the competition the predictions that were provided by our team.

**Methods**

**3.1 Data Preprocessing and Feature Engineering**
The preprocessing step is crucial for a better model prediction. The data can be inconsistent and noisy. Preprocessing helps clean the data. This improves the quality and accuracy of the training data.

In text normalization firstly, all the texts needed to be converted into lower case to achieve consistency as can be seen in the code (1.1) below.
Second step is to remove the short words that are one or two characters that are typically less informative and will not help improve our evaluation.(1.2)

Third step is removing numerical values to focus on textual content.(1.3)
Lastly special characters should be removed to retain only alphanumeric characters and spaces.(1.4)

To take out stop words, our team has imported the Natural Language Toolkit(NLTK) to remove common English stop words such as “the” ,“and”, “is”. These do not contribute significant meaning to the text. (1.5)

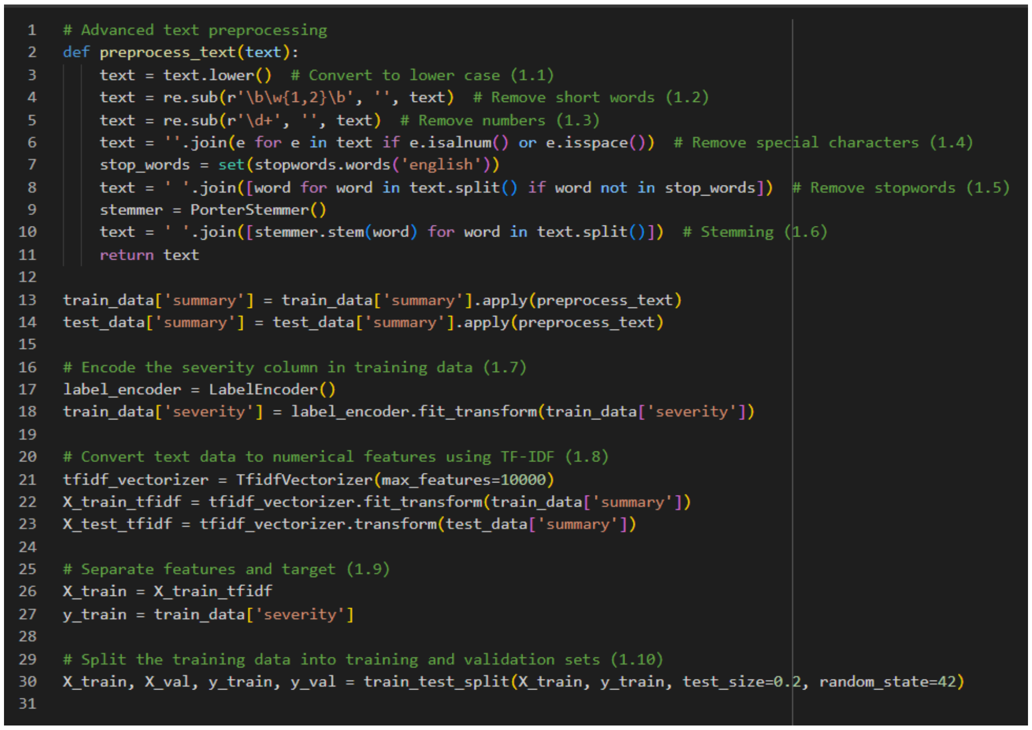
For the stemming process, we have applied the Porter Stemming algorithm to reduce words to their root, thereby consolidating variation of the same words. This enhances the performance of the text related tasks like search, text classification and sentiment analysis. (1.6)

In label encoding, the categorical target variable ‘severity’ is converted into numerical format. This transformation is necessary for a machine learning algorithm to process the target variable. (1.7)

Text vectorization uses TF-IDF which is a statistical measure to evaluate how important a word is to a document collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus.(1.8)

Feature and target separation is done for training the model of numerical features (X_train) and the target variable (y_train). (1.9)

For training and validation split, we have split the training data into two parts. One being the training data and the second is validation set which is used for evaluating the model's performance on unseen data. An 80-20 split was used, with 80% of the data for training and 20% for validation.



```python
# Advanced text preprocessing
def preprocess_text(text):
    text = text.lower()  # Convert to lower case (1.1)
    text = re.sub(r'\b\w{1,2}\b', '', text)  # Remove short words (1.2)
    text = re.sub(r'\d+', '', text)  # Remove numbers (1.3)
    text = ''.join(e for e in text if e.isalnum() or e.isspace())  # Remove special characters (1.4)
    stop_words = set(stopwords.words('english'))
    text = ' '.join([word for word in text.split() if word not in stop_words])  # Remove stopwords (1.5)
    stemmer = PorterStemmer()
    text = ' '.join([stemmer.stem(word) for word in text.split()])  # Stemming (1.6)
    return text

train_data['summary'] = train_data['summary'].apply(preprocess_text)
test_data['summary'] = test_data['summary'].apply(preprocess_text)

# Encode the severity column in training data (1.7)
label_encoder = LabelEncoder()
train_data['severity'] = label_encoder.fit_transform(train_data['severity'])

# Convert text data to numerical features using TF-IDF (1.8)
tfidf_vectorizer = TfidfVectorizer(max_features=10000)
X_train_tfidf = tfidf_vectorizer.fit_transform(train_data['summary'])
X_test_tfidf = tfidf_vectorizer.transform(test_data['summary'])

# Separate features and target (1.9)
X_train = X_train_tfidf
y_train = train_data['severity']

# Split the training data into training and validation sets (1.10)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)
```

These preprocessing steps ensure that the textual data is clean, consistent, and ready for machine learning model training, enhancing the model's ability to learn meaningful patterns from the data.

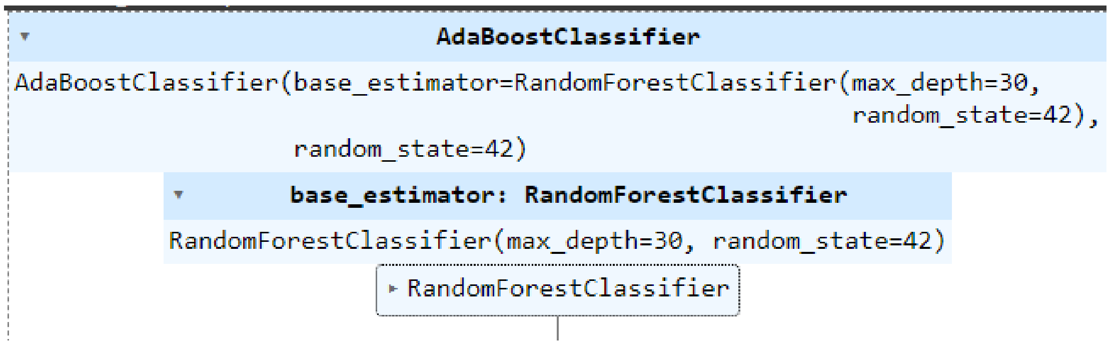
### 3.2 Model Selection

Several models were tested and fine tuned for the best performance and two were chosen:

- AdaBoost
- RandomForrest

RandomForest was chosen as a base classifier because it can handle high dimensional data which is typical in text data. RandomForest pays attention to the importance of features. RandomForest can handle both numerical and categorical data.It  can also capture non-linear relationships between features, which can be advantageous in complex sentiment classification tasks. Lastly, it is easy to tune.

Adaboost was chosen because it combines multiple weak classifiers. In this case randomforest was chosen. Adaboost can handle imbalance data which can effectively boost the performance  on minority class examples by focusing more on misclassified instances in each iteration. For the feature selection in adaboost, it can assign higher weights to more informative features. This is useful in sentiment analysis where certain keywords or phrases are strong indicators of sentiment.



```
AdaBoostClassifier
AdaBoostClassifier(base_estimator=RandomForestClassifier(max_depth=30,
                                                          random_state=42),
                   random_state=42)
base_estimator: RandomForestClassifier
RandomForestClassifier(max_depth=30, random_state=42)
RandomForestClassifier
```

### 3.3 Model Evaluation

## Results and Discussion

## Appendix