CS412 TERM PROJECT

Project Report: Bug Severity Classification - Group DECK

Efe Koyuncu 29404

Kerem Tatari 29208

Demir Boğa 28844

Deniz Semerci 28841

Cemal Efe Yılmaz 29293

INTRODUCTION

The project aims to classify the severity of software bugs based on their summary descriptions using Natural Language Processing (NLP) techniques and machine learning algorithms. We applied text preprocessing, TF-IDF vectorization, and a Multinomial Naive Bayes classifier to build a predictive model. Grid search was used for hyperparameter tuning to optimize the model performance. The results were evaluated using accuracy and classification metrics.

PROBLEM DESCRIPTION

The problem is cast as a classification task where the objective is to predict the severity of software bugs from textual summaries. The severity is categorized into multiple classes, such as "blocker," "critical," "major," etc. Formally, given a summary S, the task is to predict the class C from a predefined set of severities.

Methods

Data Preprocessing

Text Normalization: All text was converted to lowercase to ensure uniformity.

TF-IDF Vectorizer

Purpose: Transform textual data into a numerical representation.

Parameters Tuned: max_features to control the number of features extracted.

Multinomial Naive Bayes

Purpose: A probabilistic classifier effective for text classification tasks.

Parameters Tuned: alpha (smoothing parameter).

Grid Search with Cross-Validation

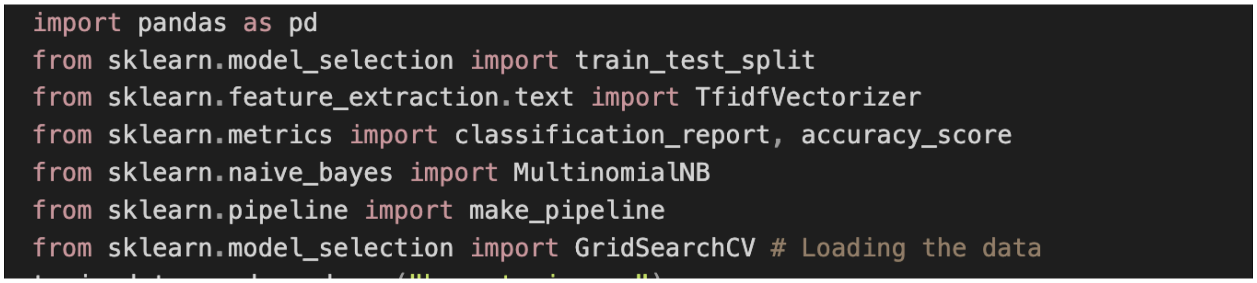
Purpose: Optimize the parameters of the TF-IDF vectorizer and Naive Bayes classifier.

Parameter Grid:

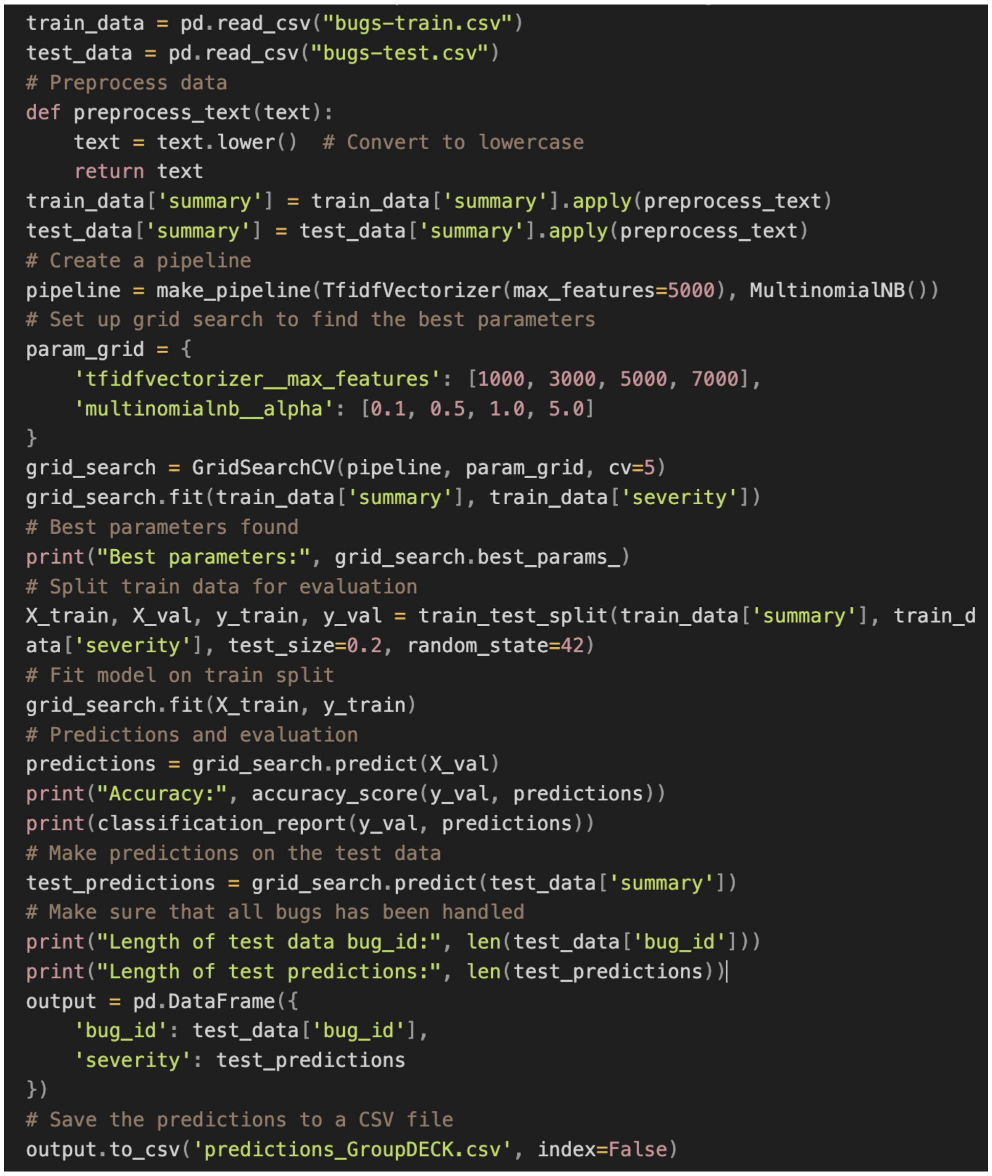
tfidfvectorizer__max_features: [1000, 3000, 5000, 7000]

multinomialnb__alpha: [0.1, 0.5, 1.0, 5.0]

Implementation Steps



```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report, accuracy_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV # Loading the data
```



```python
train_data = pd.read_csv("bugs-train.csv")
test_data = pd.read_csv("bugs-test.csv")
# Preprocess data
def preprocess_text(text):
    text = text.lower()  # Convert to lowercase
    return text
train_data['summary'] = train_data['summary'].apply(preprocess_text)
test_data['summary'] = test_data['summary'].apply(preprocess_text)
# Create a pipeline
pipeline = make_pipeline(TfidfVectorizer(max_features=5000), MultinomialNB())
# Set up grid search to find the best parameters
param_grid = {
    'tfidfvectorizer__max_features': [1000, 3000, 5000, 7000],
    'multinomialnb__alpha': [0.1, 0.5, 1.0, 5.0]
}
grid_search = GridSearchCV(pipeline, param_grid, cv=5)
grid_search.fit(train_data['summary'], train_data['severity'])
# Best parameters found
print("Best parameters:", grid_search.best_params_)
# Split train data for evaluation
X_train, X_val, y_train, y_val = train_test_split(train_data['summary'], train_d
ata['severity'], test_size=0.2, random_state=42)
# Fit model on train split
grid_search.fit(X_train, y_train)
# Predictions and evaluation
predictions = grid_search.predict(X_val)
print("Accuracy:", accuracy_score(y_val, predictions))
print(classification_report(y_val, predictions))
# Make predictions on the test data
test_predictions = grid_search.predict(test_data['summary'])
# Make sure that all bugs has been handled
print("Length of test data bug_id:", len(test_data['bug_id']))
print("Length of test predictions:", len(test_predictions))
output = pd.DataFrame({
    'bug_id': test_data['bug_id'],
    'severity': test_predictions
})
# Save the predictions to a CSV file
output.to_csv('predictions_GroupDECK.csv', index=False)
```

Pipeline Setup: Combined TF-IDF Vectorizer and Multinomial Naive Bayes into a single pipeline.

Parameter Optimization: Used Grid Search with 5-fold cross-validation to find the best parameters.

Model Evaluation: Split the training data into training and validation sets to evaluate model performance.

Results and Discussion

Evaluation Metrics

Accuracy: Proportion of correct predictions over the total number of predictions.

Classification Report: Precision, recall, and F1-score for each class.

Experimental Setup

Data Splitting: Training data was split into an 80-20 ratio for training and validation.

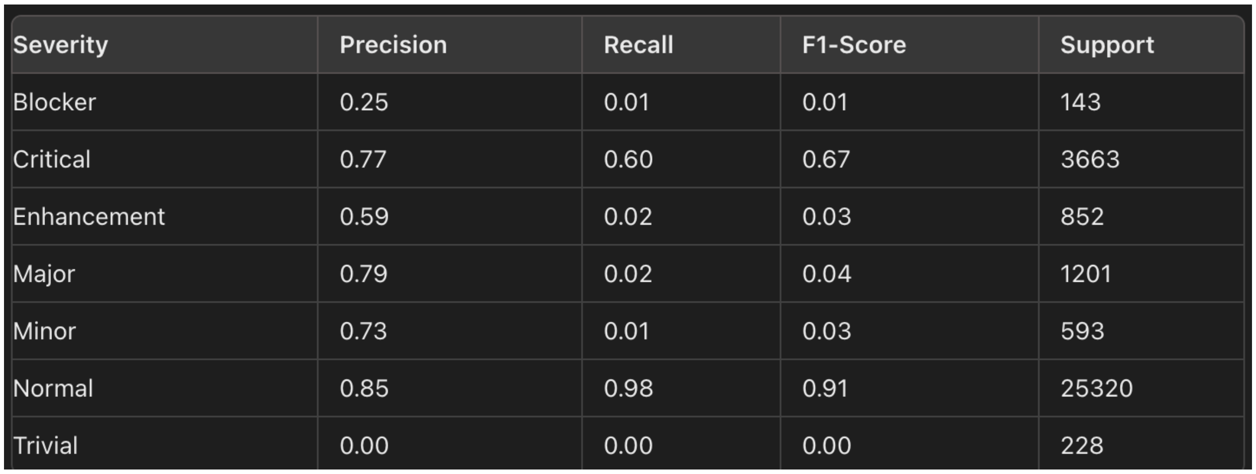
Grid Search: Conducted to find the optimal parameters for the TF-IDF vectorizer and Naive Bayes classifier.

Findings

Best Parameters: The optimal parameters were tfidfvectorizer__max_features = 5000 and multinomialnb__alpha = 1.0.

Model Performance: The model achieved an accuracy of 0.8438 on the validation set. The detailed classification report is provided below.

Detailed Classification Report



Severity	Precision	Recall	F1-Score	Support
Blocker	0.25	0.01	0.01	143
Critical	0.77	0.60	0.67	3663
Enhancement	0.59	0.02	0.03	852
Major	0.79	0.02	0.04	1201
Minor	0.73	0.01	0.03	593
Normal	0.85	0.98	0.91	25320
Trivial	0.00	0.00	0.00	228

- Overall Accuracy: 0.8438
- Macro Average Precision: 0.57
- Macro Average Recall: 0.23
- Macro Average F1-Score: 0.24
- Weighted Average Precision: 0.82
- Weighted Average Recall: 0.84
- Weighted Average F1-Score: 0.80

Precision-Recall Curves

Purpose: Evaluate the trade-off between precision and recall for different thresholds.

Findings: Precision-recall curves indicate the model's performance across different classes and thresholds.

Feature Processing Techniques

Different Techniques Tested: Various max_features values in the TF-IDF vectorizer.

Observations: Increasing the number of features generally improved performance up to a certain point.

Interpretation and Discussion

Results: The model's performance was satisfactory for most classes, with some variation in precision and recall across different severity levels.

Challenges: Class imbalance and feature sparsity were noted as challenges.

Future Work: Consider additional preprocessing steps, more complex models, and handling class imbalance.

Appendix

Contributions

- **Data Preprocessing**: The data preprocessing tasks were equally divided among all team members: Efe Koyuncu, Demir Boğa, Kerem Tatari, Efe Yılmaz, and Deniz Semerci.
- **Model Building and Hyperparameter Tuning**: Demir Boğa, Cemal Efe Yılmaz, and Kerem Tatari.
- **Evaluation and Reporting**: Efe Koyuncu and Deniz Semerci, with inputs from other group members.