# Enhanced Bug Report Classifier - User Manual

This manual provides detailed instructions on how to set up and use the Enhanced Bug Report Classifier.

## Table of Contents


1 Installation
2 Configuration
3 Data Preparation
4 Basic Usage
5 Advanced Usage
6 Command-Line Interface
7 Interpreting Results
8 Troubleshooting


## Installation

### Prerequisites

- Python 3.6 or higher
- pip package manager

### Step 1: Clone the Repository

```
git clone
https://github.com/your-username/bug-report-classifier.git
cd bug-report-classifier
```

### Step 2: Install Dependencies

Install all required packages:

```
pip install -r requirements.txt
```

### Step 3: Download NLTK Resources

The classifier requires NLTK resources for text processing. You can download them automatically by running:

```
import nltk
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
```

These will be downloaded automatically when you first run the classifier, but you can also download them manually using the above command.

### Step 4: Download Word Embeddings (Optional)

If you plan to use word embeddings, they will be downloaded automatically when first running the classifier. However, this can take some time, so you might want to download them in advance:

```
import gensim.downloader as api
```

```
api.load('glove-wiki-gigaword-100')
```

## Configuration

The classifier can be configured through various parameters:

- **Preprocessing Options**:
  - `use_stemming`: Whether to apply stemming (default: True)
  - `use_lemmatization`: Whether to apply lemmatization (default: False)
- **Feature Extraction Options**:
  - `use_embeddings`: Whether to use word embeddings (default: True)
  - `embedding_type`: Type of embeddings to use (default: 'glove-wiki-gigaword-100')
  - `tfidf_ngram_range`: Range of n-grams for TF-IDF (default: (1, 2))
  - `tfidf_max_features`: Maximum number of features for TF-IDF (default: 2000)
- **Classifier Options**:
  - `classifier_type`: Type of classifier to use, 'svm' or 'rf' (default: 'svm')
  - `verbose`: Whether to print progress information (default: True)

Each of these parameters has a significant impact on the classifier's performance:

| Parameter | Impact |
|---|---|
| `use_stemming` | Reduces words to their root form, lowering vocabulary size but potentially losing some linguistic nuance |
| `use_lemmatization` | More linguistically accurate than stemming but slower |
| `use_embeddings` | Adds semantic context but significantly increases memory usage and processing time |
| `tfidf_max_features` | Higher values capture more unique terms but increase dimensionality |
| `classifier_type` | 'svm' usually offers better performance, 'rf' might be faster for very large datasets |

## Data Preparation

The classifier expects input data in CSV format with the following columns:

- **Title**: The title of the bug report
- **Body**: The description of the bug report
- **class** or **sentiment**: The binary label (0 or 1) indicating whether the bug is performance-related

Alternatively, you can provide a CSV with a single **text** column and a **sentiment** column.

Example dataset format:

```
Title,Body,class
"GPU memory usage is very high","At the beginning of training, the
GPU memory usage is high...",1
"Error in training module","The training module fails when
processing large datasets...",0
```

If your data has a different structure, you may need to preprocess it before using the classifier.

## Basic Usage

### Using as a Module

```
from enhanced_classifier import EnhancedBugReportClassifier
# Initialize the classifier with default configuration
classifier = EnhancedBugReportClassifier()
# Load and preprocess your data
import pandas as pd
data = pd.read_csv("your_dataset.csv")
# Merge Title and Body if they exist separately
if 'Title' in data.columns and 'Body' in data.columns:
    data['text'] = data.apply(
        lambda row: row['Title'] + '. ' + row['Body'] if
pd.notna(row['Body']) else row['Title'],
        axis=1
    )
# Ensure the label column is named 'sentiment'
if 'class' in data.columns and 'sentiment' not in data.columns:
    data['sentiment'] = data['class']
# Split into train and test sets
from sklearn.model_selection import train_test_split
train_data, test_data = train_test_split(data, test_size=0.2,
random_state=42)
# Preprocess the data
train_data = classifier.preprocess_dataset(train_data)
test_data = classifier.preprocess_dataset(test_data)
# Train the classifier
classifier.train(train_data['text'].values,
train_data['sentiment'].values)
# Evaluate the classifier
metrics = classifier.evaluate(test_data['text'].values,
test_data['sentiment'].values)
# Print metrics
print(f"Accuracy: {metrics['accuracy']:.4f}")
print(f"F1 Score: {metrics['f1_score']:.4f}")
print(f"AUC: {metrics['auc']:.4f}")
```

### Using the Test Script

We provide a test script that runs a quick test on a sample of data:

```
python test_enhanced_classifier.py
```

This script will:

1. Load a sample of 100 bug reports from the PyTorch dataset
2. Split the data into training and testing sets
3. Train the classifier on the training set
4. Evaluate the classifier on the testing set
5. Print the results

## Advanced Usage

### Custom Preprocessing

You can apply custom preprocessing to your data before using the classifier:

```
# Apply custom preprocessing
from enhanced_classifier import remove_html, remove_emoji,
remove_stopwords, clean_str, apply_stemming
def custom_preprocess(text):
    text = remove_html(text)
    text = remove_emoji(text)
    text = remove_stopwords(text)
    text = clean_str(text)
    text = apply_stemming(text)
    # Add your custom preprocessing steps here
    return text
# Apply to your data
data['text'] = data['text'].apply(custom_preprocess)
# Use preprocessed data directly
X_train = train_data['text'].values
y_train = train_data['sentiment'].values
X_test = test_data['text'].values
y_test = test_data['sentiment'].values
# Train and evaluate
classifier.train(X_train, y_train)
metrics = classifier.evaluate(X_test, y_test)
```

## Grid Search for Hyperparameters

The classifier automatically performs grid search to find the best hyperparameters. You can configure the grid search by modifying the `param_grid` attribute:

```
classifier = EnhancedBugReportClassifier(classifier_type='svm')
# Customize the parameter grid
classifier.param_grid = {
    'C': [0.1, 1, 10, 100, 1000],
    'gamma': ['scale', 'auto', 0.01, 0.1, 1],
    'kernel': ['rbf', 'linear', 'poly']
}
# Train with custom grid search
classifier.train(X_train, y_train, grid_search=True, cv=5)
```

## Saving and Loading Models

To save and load trained models:

```
# Save the trained model
import pickle
# After training
with open('trained_model.pkl', 'wb') as f:
    pickle.dump(classifier, f)
# To load the model later
with open('trained_model.pkl', 'rb') as f:
    loaded_classifier = pickle.load(f)
# Use the loaded classifier
predictions = loaded_classifier.predict(test_data['text'].values)
```

## Command-Line Interface

The `run_experiments.py` script provides a command-line interface for running experiments:

```
python run_experiments.py --datasets pytorch tensorflow --repeat 5
--embeddings --classifier svm
```

Available arguments:

- `--datasets`: List of datasets to process (default: all datasets)
- `--repeat`: Number of times to repeat each experiment (default: 10)
- `--output-dir`: Directory to save results (default: 'results')
- `--stemming`: Use stemming in preprocessing (flag, default: True)
- `--lemmatization`: Use lemmatization in preprocessing (flag, default: False)
- `--embeddings`: Use word embeddings for feature extraction (flag, default: True)
- `--embedding-type`: Type of word embeddings to use (default: 'glove-wiki-gigaword-100')
- `--classifier`: Type of classifier to use, 'svm' or 'rf' (default: 'svm')
- `--ngram-min`: Minimum n-gram size for TF-IDF (default: 1)
- `--ngram-max`: Maximum n-gram size for TF-IDF (default: 2)
- `--max-features`: Maximum number of features for TF-IDF (default: 2000)
- `--verbose`: Print verbose output (flag, default: True)
- `--compare-baseline`: Compare results with baseline (flag, default: True)

Example:

```
python run_experiments.py --datasets pytorch keras --repeat 5
--classifier rf --max-features 3000
```

This will run experiments on the PyTorch and Keras datasets with a Random Forest classifier, using 3000 TF-IDF features and repeating each experiment 5 times.

## Interpreting Results

After running experiments, you'll get metrics for each dataset:

### Performance Metrics

The classifier reports the following metrics:

- **Accuracy**: Proportion of correctly classified bug reports
- **Precision**: Proportion of true positives among all positive predictions
- **Recall**: Proportion of true positives among all actual positives
- **F1 Score**: Harmonic mean of precision and recall
- **AUC**: Area Under the ROC Curve, measuring the classifier's discrimination ability

### Understanding the Results

A typical results output looks like this:

```
=== Enhanced Classifier Results for pytorch ===
Number of repeats:     10
Average Accuracy:      0.8970
Average Precision:     0.7321
Average Recall:        0.7105
Average F1 score:      0.7211
Average AUC:           0.8814
```

Here's how to interpret these numbers:

- **Accuracy**: 89.7% of bug reports were correctly classified.
- **Precision**: 73.2% of bug reports predicted as performance-related were actually performance-related.
- **Recall**: 71.1% of actual performance-related bug reports were correctly identified.
- **F1 Score**: 72.1% harmonic mean of precision and recall.
- **AUC**: 88.1% area under the ROC curve, indicating good discrimination.

### Comparison with Baseline

When the `--compare-baseline` flag is used, the results include a comparison with the baseline Naive Bayes model:

```
=== Comparison with Baseline for pytorch ===
Accuracy: 0.8669 · 0.8970 (+0.0301, +3.47%)
Precision: 0.6812 · 0.7321 (+0.0509, +7.47%)
Recall: 0.5947 · 0.7105 (+0.1158, +19.47%)
F1_score: 0.6121 · 0.7211 (+0.1090, +17.81%)
Auc: 0.8336 · 0.8814 (+0.0478, +5.74%)
```

This shows the improvement over the baseline for each metric, both in absolute terms and as a percentage.

### What to Look For

1. **F1 Score**: Since bug report classification often has class imbalance, the F1 score is a particularly important metric as it balances precision and recall.
2. **AUC**: A good indicator of the classifier's discrimination ability, especially for imbalanced classes.
3. **Improvements over Baseline**: Focus on significant improvements (>5%) over the baseline model.

## Troubleshooting

### Common Issues

1. **Missing NLTK Resources**:
   - Error: `Resource punkt not found`
   - Solution: Run `nltk.download('punkt')` to download the missing resource
2. **Memory Issues with Word Embeddings**:
   - Error: Memory error when loading word embeddings
   - Solution: Use a smaller embedding model or set `use_embeddings=False`
3. **Slow Processing Time**:
   - Issue: Classifier takes too long to train or predict
   - Solution:
     - Reduce the `tfidf_max_features` parameter
     - Set `use_embeddings=False` for faster processing
     - Use a smaller dataset for testing
4. **Dataset Format Issues**:
   - Error: Cannot find required columns

- Solution: Ensure your dataset has the required columns (Title, Body, class/sentiment, or text, sentiment)

5. **Grid Search Taking Too Long**:
   - Issue: Hyperparameter tuning takes too long
   - Solution:
     - Reduce the parameter grid
     - Set `grid_search=False` during training
     - Reduce the cross-validation folds (`cv` parameter)

## Performance Tips

1. **Balancing Speed and Accuracy**:
   - For faster processing: Disable word embeddings and use stemming
   - For better accuracy: Enable word embeddings and use a larger TF-IDF feature set
2. **Memory Optimization**:
   - Process datasets in smaller chunks if dealing with memory issues
   - Disable embeddings for very large datasets
3. **Improving Results**:
   - Try different classifier types (SVM vs. Random Forest)
   - Experiment with different preprocessing options
   - Use domain-specific stop words or custom preprocessing

## Getting Help

If you encounter issues not covered in this manual, please check the GitHub repository for updates or open an issue with the details of your problem.