Bug Report Classification Tool User Manual

This manual provides detailed instructions on how to use the Bug Report Classification tool with RoBERTa for performance-related bug classification. This tool implements the approach described in the accompanying research paper "Fine-Tuning RoBERTa for Performance-Related Bug Report Classification."

Table of Contents

Overview

The Bug Report Classification tool uses RoBERTa, a state-of-the-art transformer-based model, to classify bug reports as performance-related or non-performance-related. Key features include:

- Advanced text preprocessing for bug reports
- Fine-tuning of RoBERTa for the classification task
- Robust evaluation with multiple metrics and statistical validation
- Cross-project generalization testing
- Confusion matrix visualization

The tool significantly outperforms the baseline Naive Bayes + TF-IDF approach with substantial improvements across all evaluation metrics (accuracy, precision, recall, F1-score, and AUC).

Setup Instructions

Google Colab Setup (Recommended)

1. Upload `ISE.ipynb` to Google Colab
2. Configure runtime settings:
    - Click "Runtime" → "Change runtime type"
    - Select "GPU" as the hardware accelerator
    - Select "High-RAM" for runtime shape
    - Click "Save"
3. Run the first cell to mount Google Drive:

```
from google.colab import drive
import os

# Mount Google Drive
drive.mount('/content/drive')

# Set up project directory structure in Google Drive
BASE_DIR = '/content/drive/MyDrive/BugReportClassification'
DATA_DIR = f'{BASE_DIR}/data'
MODELS_DIR = f'{BASE_DIR}/models'
RESULTS_DIR = f'{BASE_DIR}/results'

# Create directories if they don't exist
for directory in [BASE_DIR, DATA_DIR, MODELS_DIR, RESULTS_DIR]:
    if not os.path.exists(directory):
        os.makedirs(directory)

print(f"Project directories set up at: {BASE_DIR}")
```

4. Run the second cell to install required dependencies

Local Setup

1. Clone the repository:

```
git clone https://github.com/Alva1103/ISE-Coursework.git
cd ISE-Coursework
```

2. Create and activate a virtual environment:

```
# Create virtual environment
python -m venv ise_env

# Activate environment (Windows)
ise_env\Scripts\activate

# Activate environment (Linux/Mac)
source ise_env/bin/activate
```

3. Install required packages:

```
pip install transformers==4.36.2 datasets==2.15.0 scikit-learn==1.2.2 pandas==1.5.3 numpy==1.24.0 matplotlib==3.7.2 seaborn==0.
```

4. Create the required directory structure:

```
mkdir -p BugReportClassification/data
mkdir -p BugReportClassification/models
mkdir -p BugReportClassification/results
```

5. Convert the notebook to a Python script (if needed):

```
jupyter nbconvert --to script ISE.ipynb
```

Data Preparation

Dataset Format

The tool expects CSV files with the following structure:

- `Title`: Column containing the bug report title
- `Body`: Column containing the bug report description (can be NaN)
- `class`: Binary classification label (0 for non-performance bug, 1 for performance bug)

Adding Datasets

The necessary datasets are already included in the `data/` directory of the repository:

- caffe.csv
- tensorflow.csv
- keras.csv
- pytorch.csv
- incubator-mxnet.csv

Running the Tool

Basic Usage

1. For Google Colab:
   - Run all cells in sequence using "Runtime" → "Run all" or by executing each cell individually
2. For local setup:
   - Run the Python script generated from the notebook:

```
python ISE.py
```

- Or run the notebook using Jupyter:

```
jupyter notebook ISE.ipynb
```

3. Select the project (dataset) to analyze by modifying the `project_name` variable in the final cell:

```
project_name = 'pytorch'  # Change to your preferred dataset (pytorch, caffe, tensorflow, keras, incubator-mxnet)
```

Customizing Training Parameters

You can modify the following parameters in the final cell:

```
roberta_metrics = train_roberta_model(
    data=data,
    project_name=project_name,
    epochs=10,           # Number of training epochs (default: 10)
    batch_size=16,       # Batch size for training (default: 16)
    repeat=10            # Number of experiment repeats (default: 10)
)
```

- `epochs`: Number of training epochs (default: 10)
- `batch_size`: Number of samples per batch (default: 16)
- `repeat`: Number of experiment repetitions for statistical significance (default: 10)

Interpreting Results

Performance Metrics

The tool outputs the following metrics for each run and as averages across all repeats:

- **Accuracy**: Percentage of correctly classified reports
- **Precision (macro)**: Ability to avoid false positives
- **Recall (macro)**: Ability to find all positive instances
- **F1 Score (macro)**: Harmonic mean of precision and recall
- **AUC**: Area under the ROC curve, measuring discriminative ability

Expected Performance

Based on the research paper, you should expect the following performance improvements over the baseline:

Metric	Naive Bayes + TF-IDF	RoBERTa (Our Model)	Improvement
Accuracy	0.5747	0.9155	+59.30%
Precision	0.6082	0.8210	+34.99%
Recall	0.7066	0.8297	+17.42%
F1-Score	0.5240	0.8196	+56.41%
AUC	0.7066	0.9202	+30.23%

Cross-Project Performance

When testing the model's ability to generalize across different projects (training on four frameworks and testing on the fifth), expect F1-scores similar to:

Test Project	Naive Bayes + TF-IDF	RoBERTa (Ours)	Improvement
TensorFlow	0.5406	0.8916	+64.93%
PyTorch	0.5519	0.7860	+42.42%
Keras	0.5369	0.8449	+57.37%
MXNet	0.5479	0.8533	+55.74%
Caffe	0.4428	0.7221	+63.08%

Visualization

The tool generates confusion matrices for the final repeat of each experiment, showing:

- True Positives (TP)
- False Positives (FP)
- True Negatives (TN)
- False Negatives (FN)

These visualizations help identify which types of errors are most common in the model's predictions.

Model Architecture

Overview

The tool uses the following architecture, as described in the research paper:

1. **Input Processing**:
    - Bug report titles and descriptions are combined
    - Text is preprocessed (HTML removal, emoji removal, stopword removal, normalization)
    - Input is tokenized and truncated to 512 tokens
2. **RoBERTa Encoder**:
    - Pre-trained roberta-base model transforms tokens into contextual embeddings
3. **Classification Head**:
    - The [CLS] token embedding is passed through a fully connected layer with softmax activation
    - Output is binary classification (0/1)

Training Procedure

- **Optimizer**: AdamW with learning rate 2e-5, epsilon 1e-8
- **Learning Rate Scheduler**: Linear schedule with warmup
- **Loss Function**: Cross-entropy loss
- **Gradient Clipping**: Applied at 1.0

Advanced Usage

Text Preprocessing Customization

The preprocessing pipeline can be customized by modifying the functions in the "Define Text Preprocessing Methods" cell:

```
def remove_html(text):
    """Remove HTML tags using a regex."""
    html = re.compile(r'<.*?>')
    return html.sub(r'', text)

def remove_emoji(text):
    """Remove emojis using a regex pattern."""
    emoji_pattern = re.compile("["
                               u"\U0001F600-\U0001F64F"  # emoticons
                               u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                               u"\U0001F680-\U0001F6FF"  # transport & map symbols
                               u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                               u"\U00002702-\U000027B0"
                               u"\U000024C2-\U0001F251"
                               "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', text)

# Add or modify stopwords
custom_stop_words_list = ['...'] # Add your custom stopwords here
```

Troubleshooting

Common Issues

1. **CUDA Out of Memory Error**:
   - Reduce batch size (try 8 or 4 instead of 16)
   - Reduce maximum sequence length in the BugReportDataset class (try 256 instead of 512)
   - Use a smaller model variant if available
2. **Training Too Slow**:
   - Reduce number of repeats for testing (try 2 instead of 10)
   - Reduce number of epochs (try 4 instead of 10)
   - Ensure GPU is being utilized (check device output)
3. **Results Different from Paper**:
   - Verify all random seeds are set to 42
   - Ensure exact package versions match those specified
   - Check for preprocessing differences
4. **Data Loading Issues**:
   - Ensure CSV files have the correct columns ('Title', 'Body', 'class')
   - Check for encoding issues in CSV files (use UTF-8 encoding)
   - Verify file paths are correct for your environment

Google Colab-Specific Issues

1. **Session Timeout**:
   - Enable "Settings" → "Miscellaneous" → "Receive notification when GPU is available" to avoid losing progress
   - Consider saving intermediate results to Google Drive
2. **Google Drive Mounting Failure**:
   - Re-run the mounting cell
   - Check if you have authorized access to your Google Drive

Local Setup Issues

1. **CUDA Not Available**:
   - Verify PyTorch installation with CUDA support:

     ```
     import torch
     print(torch.cuda.is_available())
     print(torch.cuda.get_device_name(0) if torch.cuda.is_available() else "No GPU")
     ```

   - Install the correct version of PyTorch for your CUDA version from https://pytorch.org/ (https://pytorch.org/)
2. **Package Compatibility Issues**:
   - Create a clean virtual environment
   - Install packages in the order specified in the requirements

For additional assistance, please refer to the GitHub repository's issues section or contact the authors.