User Manual - Bug Report Classification Tool

This tool is designed to classify bug reports from deep learning frameworks as either performance-related or not. It implements three different classification models with increasing sophistication:

1. **Baseline Model**: Naive Bayes with TF-IDF features
2. **Intermediate Model**: SVM with Word2Vec embeddings
3. **Hybrid Model**: Ensemble classifier with domain-specific features, pattern detection, and code-aware analysis

Getting Started

1. Ensure all requirements are installed (see `requirements.pdf`)
2. Place your datasets in the `datasets` directory
3. Run the download_nltk_resources.py script to get necessary NLTK data

Tool Structure

The tool consists of several key components:

Core Components

- **Model Implementations**:
  - `baseline_model.py` - Naive Bayes classifier with TF-IDF features
  - `intermediate_model.py` - SVM classifier with Word2Vec word embeddings
  - `hybrid_model.py` - Advanced ensemble model with domain-specific features
- **Preprocessing and Feature Extraction**:
  - `preprocessing.py` - Text cleaning with special handling for code and technical terms
  - `feature_extraction.py` - TF-IDF, Word2Vec, and structural feature extraction
- **Evaluation Framework**:
  - `evaluation.py` - Comprehensive evaluation metrics and visualisation
  - `evaluation_framework.py` - Framework for evaluating models across datasets

Testing Scripts

- `test_all_models.py` - Compare all three models
- `test_intermediate_model.py` - Test baseline vs. intermediate model
- `test_hybrid_model.py` - Test baseline vs. hybrid model
- `test_framework.py` - Test the framework components

Models Description

Baseline Model

- **Algorithm**: Multinomial Naive Bayes
- **Features**: TF-IDF features from bug report text
- **Advantages**: Fast training and prediction, works well with text classification
- **Class balancing**: SMOTE for handling class imbalance

Intermediate Model

- **Algorithm**: Support Vector Machine (SVM)
- **Features**: Word2Vec embeddings (averaged word vectors)
- **Advantages**: Better semantic understanding of text
- **Class balancing**: SMOTE for handling class imbalance

Hybrid Model

- **Algorithm**: Ensemble of multiple classifiers (Naive Bayes, SVM, Random Forest, Logistic Regression)
- **Features**:
  - Framework-specific weighted TF-IDF
  - Pattern-based features using regex for performance indicators
  - Code-aware token extraction
  - Meta-features about report structure
- **Advantages**: Superior performance through domain knowledge and specialised feature engineering
- **Class balancing**: SMOTE plus weighted classification

Running the Classification

Test on a Single Framework

To evaluate models on a specific framework, use (make sure you are the in the lab1 folder - cd lab1)

```
# Test hybrid model vs baseline on TensorFlow
python test_hybrid_model.py --framework tensorflow

# Test intermediate model vs baseline on PyTorch
python test_intermediate_model.py --framework pytorch
```

Test on All Frameworks

To run evaluation across all available frameworks:

```
# Evaluate all models on all datasets
python evaluation_framework.py

# Evaluate with reduced sample size for faster results
python evaluation_framework.py --sample_ratio 0.2 --n_runs 2

# Test hybrid model vs baseline on all frameworks
python test_hybrid_model.py

# Test intermediate model vs baseline on all frameworks
python test_intermediate_model.py

# Test all models on all frameworks with results in evaluation_summary.md (will take
        longer time) for full comparison
python test_all_models.py
```

Parameters

- `--framework`: Specify framework dataset to use (tensorflow, pytorch, keras, caffe, all)
- `--n_runs`: Number of evaluation runs for statistical validity (default: 3)
- `--test_size`: Percentage of data to use for testing (default: 0.3)
- `--sample_ratio`: Portion of dataset to use for faster evaluation (default: 0.5)
- `--output_dir`: Directory to save results (default: ./results)

Output Files and Visualisation

The tool generates various output files in the `results` directory:

Results Directory

- CSV files with metrics for individual runs
- `*_multiple_runs.csv` files containing results from multiple evaluation runs
- `evaluation_summary.md` with overall metrics across frameworks

Plots Directory

These visualisations help understand model performance:

1. **F1 Score Comparison** (`*_f1_score_comparison.png`):
   - Bar charts comparing F1 scores across models
   - Higher bars indicate better overall performance
2. **Precision-Recall Comparison** (`*_precision_recall.png`):
   - Shows precision and recall for each model
   - Ideal models have both high precision and recall
3. **Boxplots** (`*_f1_score_boxplot.png`):
   - Show statistical distribution of F1 scores across multiple runs
   - Wider boxes indicate more variability in performance
4. **Training Time Comparison** (`*_training_time.png`):
   - Bar charts comparing training times
   - Faster training times indicate better efficiency

Comprehensive Results Directory

The `comprehensive_results` directory contains: - Detailed tables for each framework - Raw data for all evaluation metrics - Cross-framework comparison summaries - Model parameter settings used for each run

Interpreting evaluation_summary.md

The `evaluation_summary.md` file presents an overview of all model performances:

1. **Framework-specific sections** show metrics for each deep learning framework:
   - **Precision**: Higher values mean fewer false positives
   - **Recall**: Higher values mean fewer false negatives
   - **F1 Score**: Harmonic mean of precision and recall (balance between the two)
   - **Training/Prediction Time**: Resource requirements for each model
2. **Overall Model Performance** section provides average performance across all frameworks
   - Values are shown as mean ± standard deviation
   - Higher precision, recall, and F1 scores indicate better performance
3. **Note on missing models**: Sometimes models may not appear in some result files
   - The Hybrid model may be missing if memory resources were limited
   - Only completed evaluations are included in the summaries

Working with New Datasets

To use your own bug report datasets:

1. Format your data as CSV files with at least these columns:

    - 'Title': Bug report title
    - 'Body': Bug report description
    - 'class': Binary label (1 for performance bug, 0 for non-performance bug)

2. Place your CSV files in the `datasets` directory with framework name as filename (e.g., `tensorflow.csv`)

3. Run the evaluation as described above

Reducing Evaluation Time

If the full evaluation takes too long, you can:

1. Test on a single framework:

```
python test_hybrid_model.py --framework tensorflow
```

2. Reduce the sample size:

```
python evaluation_framework.py --sample_ratio 0.2
```

3. Reduce the number of statistical runs:

```
python evaluation_framework.py --n_runs 2
```