Intel® Unnati Industrial Training 2025

Problem Statement – 01 :Bug Detection and Fixing - Project

Documentation

Introduction

This project consists of two Jupyter Notebook files implementing a machine learning algorithm for bug detection and fixing. The project includes:
1. Data Extraction & Preparation
2. Bug Detection using a fine-tuned LLM model

1. Data Extraction & Preparation

Data Source & Collection

- The dataset is scraped from GitHub using the Developer API.
- PR (Pull Request) data was collected and labeled as 'buggy,' 'bug-free,' or 'fixed.'
- Over 100,000 commits were analyzed, and the most relevant ones were saved as 'github_scrape.csv.'
- A Kaggle dataset could have been used, but GitHub's collaborative nature provides richer technical insights.

Data Analysis & Storage

- The scraped dataset underwent analysis and was stored as a CSV file.
- This process was implemented in 'Data_Scraper_github.ipynb.'
- A sample GitHub API token (valid for one month) was used for testing.

2. Bug Detection Project

The 'bug_detection.ipynb' file implements a fine-tuned LLM (NLP) model for detecting and fixing bugs.

Steps in Implementation:

Step 1: Environment Setup

- The project is designed to run on Google Colab for platform independence.
- Required Python packages are installed at the start.
- For local execution, a virtual environment is recommended.

Step 2: Importing Dependencies
- Essential libraries and models are imported.

Step 3: Data Loading
- The dataset ('github_scrape.csv') is uploaded manually in Google Colab.
- For local execution, the dataset path is initialized.

Step 4: Data Processing
- Pandas is used to load the CSV file into a DataFrame.
- The dataset is structured with 'code' and 'label' columns.

Step 5: Data Splitting
- The dataset is split into training (90%) and validation (10%) sets.

Step 6: Converting Data for Model Training
- Training and validation DataFrames are converted into Hugging Face datasets.

Step 7: Model Selection
- A BERT-like model is chosen for fine-tuning.
- The 'microsoft/codebert-base' model is used.
- Tokenization is applied to train and validation datasets.
- Labels are mapped to 'bug-free' and 'buggy'.

Step 8: Performance Metrics
- The model is evaluated based on:
  - Accuracy
  - Precision
  - Recall
  - F1 Score

Step 9: Training the Model
- Training arguments are set up.
- Model training is conducted using Weights & Biases (wandb) integration.
- Training requires 3-4 hours due to the large dataset.

Step 10: Syntax Error Detection
- A classification script is implemented to parse code and detect syntax errors.

Step 11: Bug Fixing
- CodeT5 is used to automatically generate fixes for detected bugs.

Step 12: Testing the Model

- The model is tested with buggy inputs.
- It successfully detects bugs, provides detailed explanations, and suggests fixes.

Tools Used

- Google Colab
- Jupyter Notebook
- Machine Learning Models
- Weights & Biases (wandb)
- Hugging Face
- GitHub API
- GCP - E2 Cluster Instance

Conclusion

This project successfully implements a machine learning model to detect bugs in Python code and provide possible solutions. The approach integrates GitHub data, fine-tuned models, and NLP techniques to enhance bug detection and fixing capabilities.

License

Author Details

Author: Debayan Ghosh
Student Code: BWU/BTA/22/157
Problem Statement 01: Bug Detection and Fixing