Problem Statement

This assignment aimed to improve the existing machine learning model for identifying file languages. The original model could classify only three languages (C++, Java, and Python). My goal was to expand its capabilities while ensuring efficient performance on limited hardware resources.

Approach

1. Expanding Language Classification

I modified the provided code (*pred_lang_io.py*) to train the model on a dataset containing eight languages (C++, Java, Python, Groovy, JavaScript, XML, JSON, and YAML). The model successfully learned to classify these languages. The detailed evaluation metrics, including the confusion matrix and F1 scores, are available in the "8_classes_prediction_result.txt" file.

While the F1 scores were acceptable, there was still room for improvement.

2. Improving Model Accuracy and Scalability

The initial solution involved modifying the existing model's hidden layers (increasing the number of layers and neurons). However, this approach presented scalability challenges. Adding additional languages would require further adjustments to the hidden layers, leading to a time-consuming trial-and-error process.

To address this, I explored pre-trained models. These models are already trained on a massive amount of data and can be fine-tuned for specific tasks like this. I chose two pre-trained models: "bert-tiny" and "bert-mini" due to their smaller size, which translates to faster training and inference times. This aligns with the hardware constraints (single quad-core CPU and 512MB RAM) for achieving the target processing speed of 4 files per second.

The code used for training these Bert models is in the *bert_training.py* file and the throughput testing code is in *inference_test.py* file.

Findings

The fine-tuned models' performance is documented in "bert-tiny_metrics.txt" and "bert-mini_metrics.txt" respectively. I observed that the F1 scores for most languages improved significantly compared to the original model. Notably, "bert-mini" achieved the best overall performance while maintaining quick training and inference speeds. It achieved over **0.97 F1-score** for every class except for the groovy class while the throughput of the model was **15 files/second**.

1. **Addressing Groovy Class Performance**

Although "bert-mini" performed well for most languages, it struggled with the Groovy class. After analyzing the results. I realized that the model had difficulty because there was less training data for Groovy compared to other languages. To improve the model's performance with Groovy, I adjusted the weight_decay parameter from **0.01 to 0.05**. This change increased its F1-score from **0.85 to 0.93**.

Potential for Further Improvement

Given more time, I would focus on enhancing the model's performance for the Groovy class. Two potential strategies could be explored:

1. **Data Augmentation:** Techniques like SMOTE can be employed to artificially create additional Groovy data points. This would provide the model with more context and improve its ability to recognize Groovy code patterns.
2. **Acquiring Additional Data:** Obtaining a larger dataset with a more balanced representation of all languages would benefit the model's overall performance.

By implementing one or both of these approaches, I expect to see a significant boost in the model's accuracy for classifying the Groovy language and possibly other underrepresented languages in future deployments.

How is this approach scalable and better?

1. Ease of Adding More Classes:

BERT's architecture is designed to handle a broad range of language tasks. BERT's contextual understanding helps it adapt to new classes with relatively minimal effort. So making this model support newer languages in the future will always be easier.

2. Scalability:

BERT models are highly scalable due to their transformer architecture, which is efficient in handling large datasets and high-dimensional data. They can be fine-tuned on large corpora and can be easily distributed across multiple GPUs for faster processing. Their scalability makes them suitable for large-scale applications.