A Recurrent Neural Network Pipeline for Programming Language Prediction

Thivina Suduwa Devage
Student Number : 1008125360, E-mail : thivina.suduwadevage@mail.utoronto.ca
https://github.com/Thivina-Hemarathna/RNN-Pipeline-for-Programming-Language-Detection.git

Introduction

Identifying the programming language of a source code snippet is a fundamental task in software engineering, with applications in code search, repository mining, automated documentation, plagiarism detection, and developer productivity tools. While simple heuristics such as file extensions or keyword matching can work in constrained settings, they often fail on short, incomplete code fragments commonly found in forums, logs, and educational submissions. The goal of this project is to build a robust deep learning system that predicts the programming language of a given code snippet directly from its raw textual representation.

This project proposes a Recurrent Neural Network (RNN)–based pipeline that models code as a sequence of characters or tokens and learns language-specific syntactic and stylistic patterns. Programming languages exhibit strong sequential structure (e.g., keyword ordering, punctuation usage, indentation patterns), making sequence models a natural choice. Deep learning is particularly suitable for this task because it can automatically learn discriminative representations without relying on hand-crafted rules, and it scales well to large and diverse programming languages.

Illustration

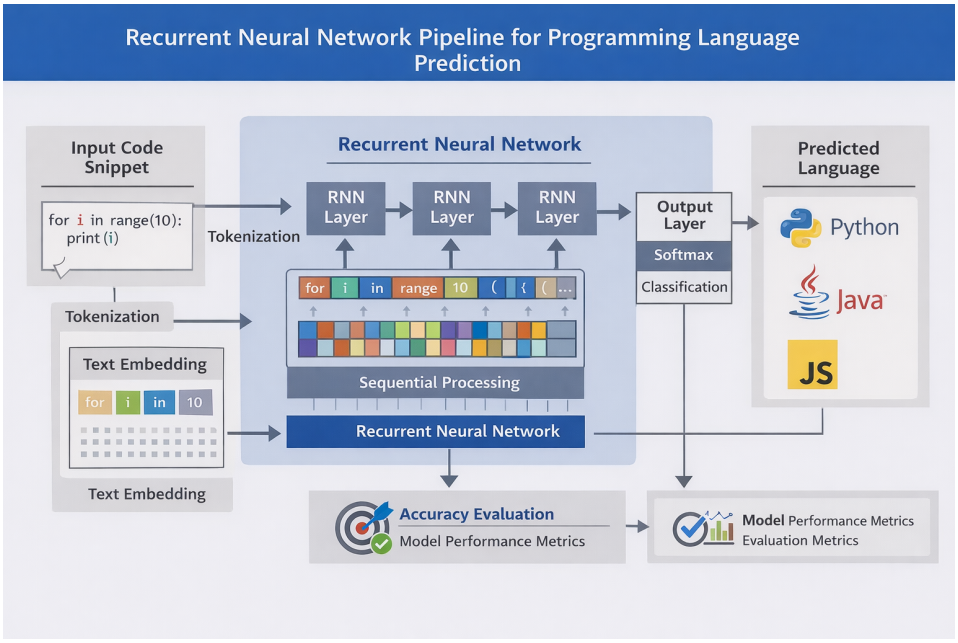

Figure 1: Overall pipeline for programming language prediction using an RNN-based model.

The above Figure 1 illustrates the overall pipeline of the proposed system. Raw source code snippets are first preprocessed and converted into sequences of characters or tokens. These sequences are

embedded into dense vector representations and passed through an RNN encoder (e.g., LSTM or GRU). The final hidden representation is then fed into a fully connected classifier that outputs a probability distribution over programming language labels.

Background & Related Work

Programming language identification has been studied using both traditional machine learning and deep learning approaches. Pritchard et al. [1] explored n-gram models and classical classifiers for source code classification. The Pygments library [2] uses rule-based lexical analysis for language detection, which is effective but brittle to noise.

Recent work has shifted toward neural models. Lopez et al. [3] demonstrated that character-level RNNs can effectively classify code snippets across dozens of languages. Karampatsis and Sutton [4] showed that neural language models can capture rich syntactic patterns in code. Transformers have also been applied to source code understanding, such as CodeBERT [5], though they are computationally more expensive. This project builds on these ideas while focusing on a simpler and more interpretable RNN-based pipeline.

Data Processing

The primary dataset for this project will be collected from public GitHub repositories and existing curated datasets such as the CodeSearchNet corpus [6]. Code files from multiple programming languages (e.g., Python, Java, C++, JavaScript, and Go) will be extracted and segmented into smaller snippets of fixed or variable length.

Preprocessing steps include removing comments where appropriate, normalizing whitespace, and filtering out extremely short or non-informative snippets. Depending on the experimental setup, code will be represented either as character sequences or token sequences produced by a lightweight lexer. Each snippet will be labeled by its ground-truth programming language, and the dataset will be split into training, validation, and test sets.

Architecture

The core model is a Recurrent Neural Network encoder, implemented using either Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) cells. Input sequences are first mapped to embeddings, which are then processed sequentially by the RNN. The final hidden state (or an aggregation of hidden states) is used as a fixed-length representation of the code snippet. This representation is passed to a fully connected layer with softmax activation to produce language predictions.

Baseline Model

As a baseline, a traditional machine learning classifier will be implemented using character-level n-gram features combined with a multinomial Naive Bayes or Support Vector Machine (SVM) classifier. This baseline is simple, well-understood, and commonly used for text classification tasks, providing a meaningful point of comparison against the RNN-based model.

Ethical Considerations

The dataset is sourced from publicly available code repositories; however, care must be taken to respect licensing terms and avoid misuse of proprietary code. The model may also inherit biases from the dataset, such as over-representing popular languages or coding styles. Additionally, language prediction models should not be used as the sole basis for plagiarism accusations or automated grading decisions without human oversight.

References

[1] Pritchard, D., Becker, B., and Ostermann, K. A comparative study of programming language classification techniques. *Proceedings of the ACM Conference on Programming Education*, 2019.

[2] Pygments Development Team. Pygments: A generic syntax highlighter. `https://pygments.org/`.

[3] Lopez, J., Ram'irez, A., and Torres, M. Automatic programming language identification using neural networks. *IEEE Access*, 2020.

[4] Karampatsis, R. M., and Sutton, C. Maybe deep neural networks are the best choice for modeling source code. *Proceedings of EMNLP*, 2020.

[5] Feng, Z., et al. CodeBERT: A pre-trained model for programming and natural languages. *Findings of EMNLP*, 2020.

[6] Husain, H., et al. CodeSearchNet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*, 2019.