PHÂN LOẠI ĐA NHÃN TRONG LẬP TRÌNH THI ĐẤU DỰA VÀO BIỂU DIỄN CPG

MULTI-LABEL CLASSIFICATION IN COMPETITIVE PROGRAMMING BASED ON CPG REPRESENTATION

SVTH: Phan Văn Hưng

Lớp 20TCLC_DT5, Khoa Công Nghệ Thông Tin, Trường Đại học Bách khoa Đà Nẵng; Email: pvhung1302@gmail.com

GVHD: Phạm Minh Tuấn

Khoa Công Nghệ Thông Tin, Trường Đại học Bách khoa Đà Nẵng; Email: pmtuan@dut.udn.vn

Tóm tắt - Một trong những cách tốt nhất để các lập trình viên cải thiện kĩ năng của họ một cách thú vị và đầy thử thách là giải quyết các vấn đề trong lập trình thi đấu. Đối với những người chưa có kinh nghiệm, một số vấn đề có thể tỏ ra khó khăn, đòi hỏi một số gợi ý để lập trình giải pháp. Vì vậy, nhiệm vụ của nghiên cứu này tập trung vào việc tự động hoá gắn nhãn thuật toán hay hướng tiếp cận mô tả vấn đề lập trình bằng cách sử dụng phương pháp học sâu. Chúng tôi nhận thấy rằng việc biểu diễn mã nguồn hiệu quả là điều cần thiết trong các nhiệm vụ như phân loại mã nguồn hay phát hiện bản sao của mã nguồn,… Do đó, chúng tôi đã xây dựng mô hình mạng neuron sử dụng cơ chế chú ý với đầu vào là tập hợp các đường dẫn AST, CFG, PDG. Tập hợp các đường dẫn đó được gọi là đồ thị biểu diễn thuộc tính mã nguồn (Code Property Graph – CPG).

Từ khóa – Xử lí ngôn ngữ tự nhiên; Đồ thị cấu trúc mã nguồn; Phân loại đa nhãn; Phân loại mã nguồn; Biểu diễn mã nguồn.

Abstract – Competitive programming is one of the best ways for developers to grow their skills in a fun and challenging way. Some problems could seem overly difficult to the inexperienced person, requiring some hints to implement the solution. Therefore, this paper aims to use deep learning to automatically label the algorithm or approach that describes the programming problem. We discovered that effective source code representation is crucial for various software engineering tasks like code classification and code clone detection,... Thus, we created the attention-based neural network model with input that is the aggregated representation including AST, CFG, and PDG. That aggregated representation is called Code Property Graph (CPG).

Key words - Natural Language Processing; Code Property Graph; Multi-label classification; Code Classification; Code Presentation.

1. Đặt vấn đề

1.1. Ứng dụng

Những năm gần đây, ngày càng có nhiều người quan tâm đến lập trình thi đấu, hoặc vì mục đích giáo dục đơn thuần hay để chuẩn bị cho các cuộc phỏng vấn xin việc. Theo xu hướng này, chúng ta cũng có thể thấy sự gia tăng số lượng các nền tảng trực tuyến cung cấp các dịch vụ như giải quyết vấn đề lập trình hoặc hướng dẫn lập trình. Những bài toán lập trình này dựa trên một số lượng chiến lược có hạn. Nắm bắt được chiến lược nào để giải quyết bài toán đó là chìa khoá để thiết kế giải pháp. Tuy nhiên, nó rất khó để suy luận trực tiếp từ văn bản câu hỏi, đặc biệt là những người với ít hoặc không có kinh nghiệm trong lập trình thi đấu. Do đó, việc hỗ trợ lập trình viên xác định được các chiến lược khả thi có thể tỏ ra rất hữu ích, hiệu quả và mang tính giáo dục.

Để đạt được điều này, chúng tôi đề xuất một hệ thống tự động gắn nhãn một vấn đề lập trình dựa trên các mã nguồn đã giải quyết nó. Các nhãn này có thể chung chung, chẳng hạn như "Math" hay "Implementation" hoặc cụ thể hơn như "Binary Search", "Combinatorics". Mỗi vấn đề có thể có nhiều nhãn, vì vậy nghiên cứu này tập trung vào bài toán phân loại đa nhãn, là một vấn đề khó khăn hơn so với phân loại đa lớp thông thường, trong đó mỗi điểm dữ liệu chỉ có thể có một nhãn đính kèm. Một vấn đề tương tự là dự đoán các nhãn này dựa trên văn bản câu hỏi, nhưng điều này sẽ có hạn chế bởi vì mâu thuẫn ngôn ngữ. Vì vậy, phần đầu vào của hệ thống là các mã nguồn.

```
bool even_num(int x){
    if (x % 2 ==0) return true;
    else return false;
}
```

Labels: Math

Hình 1: *Ví dụ về 1 mã nguồn*

1.2. Các nghiên cứu hiện nay

Do sự tiến bộ của các phương pháp phân tích và những kiến trúc học sâu, các nhà nghiên cứu đã tập trung trong việc phân tích ngữ nghĩa của cú pháp, trích xuất các thuộc tính của chương trình một cách tự động để làm cho phần mềm trở nên dễ hiểu và dễ bảo trì hơn. Vì vậy, biểu diễn mã nguồn là trọng tâm và có ảnh hưởng lớn đến hiệu suất của các hướng tiếp cận.

Nhiều nghiên cứu hiện nay đã sử dụng biểu diễn cây và đồ thị như AST, CFG hoặc PDG trong việc biểu diễn mã nguồn. Trong đó, code2vec [1] đã sử dụng đường dẫn AST với mô hình mạng neuron sử dụng cơ chế chú ý và ứng dụng được trong nhiều công việc. Hay đồ thị CPG đã được Yamaguchi [3] sử dụng để biểu diễn mã nguồn trong việc tìm lỗ hổng phần mềm. Hơn nữa, mocktail [2] đã thực hiện thử nghiệm cho thấy rằng các biểu diễn ngữ nghĩa CFG, PDG,… cải thiện hiệu suất mô hình.

Vì vậy, những đóng góp mà bài báo của chúng tôi đem lại gồm:

- Sử dụng biểu diễn CPG để biểu diễn mã nguồn, làm đầu vào cho mô hình.
- Xây dựng mô hình phân loại nhãn thuật toán hay cách tiếp cận các vấn đề lập trình thi đấu.

Phần còn lại của bài báo bao gồm: Phần 2 trình bày lý thuyết nghiên cứu. Phần 3 trình bày tổng quan bộ dữ liệu.

Phần 4 trình bày phương pháp phân loại đa nhãn. Phần 5 trình bày kết quả thực nghiệm của mô hình. Phần 6 trình bày kết luận.

2. Cơ sở lý thuyết

2.1. Biểu diễn mã nguồn

Các nhà nghiên cứu về phân tích chương trình và thiết kế trình biên dịch đã phát triển nhiều trong biểu diễn mã nguồn trung gian. Các cách biểu diễn này biểu thị rất tốt các thuộc tính của chương trình. Trong nghiên cứu này, chúng tôi sử dụng ba trong số cách biểu diễu nổi bật. Đó là cây cú pháp trừu tượng (Abstract Syntax Tree - AST), đồ thị luồng điều khiển (Control Flow Graph – CFG) và đồ thị phụ thuộc chương trình (Program Dependence Graph – DPG). Chúng tôi định nghĩa chúng lần lượt:

Định nghĩa 1 (Abstract Syntax Tree): là một cây có giới hạn và có định hướng. Đây là cấu trúc cây mà các nút gốc được gán nhãn bằng các toán tử và các nút con được gán nhãn bằng các toán hạng hoặc biến. AST biểu diễn các cú pháp trong mã nguồn, nó trừu tượng bởi vì nó không thật sự biểu diễn hoàn toàn cú pháp mà chỉ là cấu trúc của chúng.

Định nghĩa 2 (Control Flow Graph): là đồ thị có hướng, trong đó các nút thể hiện vị ngữ hoặc câu lệnh và các cạnh miêu tả con đường dòng điều khiển giữa các nút. Nó mô tả thứ tự thực thi các câu lệnh trong chương trình.

Định nghĩa 3 (Program Dependence Graph): Là 1 đồ thị có hướng, trong đó các nút thể hiện vị ngữ hoặc câu lệnh và các cạnh liên kết chúng thể hiện sự phụ thuộc trong điều khiển và dữ liệu. Vì vậy, cạnh của đồ thị PDG bao gồm 2 loại: cạnh phụ thuộc điều khiển thể hiện kết quả của một vị từ ảnh hưởng đến biến và cạnh phụ thuộc dữ liệu thể hiện kết quả của một biến này ảnh hưởng đến một biến khác.

2.2. Cơ chế chú ý

Cơ chế chú ý (Attention Mechanism [8]) trong học sâu là kĩ thuật nhấn mạnh những đầu vào giúp cho mô hình dự đoán tốt hơn, đồng thời ít chú ý vào những đầu vào không giúp. Lớp Attention thực hiện được điều này bằng cách gán trọng số trên các đầu vào để tính toán trọng số trung bình của tất cả các đầu vào. Ưu điểm chính của mô hình năm bắt được sự khác biệt trong các đầu vào tương tự thông qua sự khác biệt trọng số được gán cho chúng.

Cho $x_1, x_2, \dots, x_n$ là đầu vào của lớp Attention. Mục đích của lớp Attention là học được vector Attention a, được sử dụng để tính toán trọng số α_i của x_i. Cho Vector Attention a , trọng số w_i và vector trọng số trung bình $\tilde{w}$ được tính toán như sau:

$$w_i = \frac{exp(a^T \cdot x_i)}{\sum_{j=1}^{n} exp(a^T \cdot x_j)} \quad (1)$$

$$\tilde{w} = \sum_{i=1}^{n} \alpha_i \cdot x_i \quad (2)$$

Vector trọng số trung bình $\tilde{w}$ thể hiện một biểu diễn duy nhất cho bộ dữ liệu đầu vào (Eq 2).Vector Attention a sẽ được học theo thời gian để tạo ra một biểu diễn tốt hơn cho tập dữ liệu. Loại cơ chế chú ý này được gọi là cơ chế chú ý Luong.

3. Bộ dữ liệu

3.1. Thu thập dữ liệu

Vấn đề chính mà chúng tôi gặp phải khi xây dựng thuật toán là tìm kiếm dữ liệu mã nguồn từ các nền tảng trực tuyến về lập trình. Đặc biệt, nguồn dữ liệu phải đáp ứng những yêu cầu sau:

- Số lượng lớn vấn đề lập trình
- Mã nguồn phải chính xác ứng với vấn đề lập trình.
- Được gán nhãn thuật toán hay cấu trúc dữ liệu có độ chính xác cao
- Mã nguồn được lập trình bởi ngôn ngữ bậc cao C++.

Một nền tảng đã đáp ứng được những yêu cầu trên là CodeForces, một nền tảng trực tuyến nổi tiếng về lập trình thi đấu. Đồng thời, chúng tôi đã thu thập mã nguồn từ những bộ dữ liệu có sẵn như AlgoLabel [4], Competitive Programming Problem Tag Classification [5], có chứa những mã nguồn từ nền tảng CodeForces.

3.2. Tiền xử lí dữ liệu

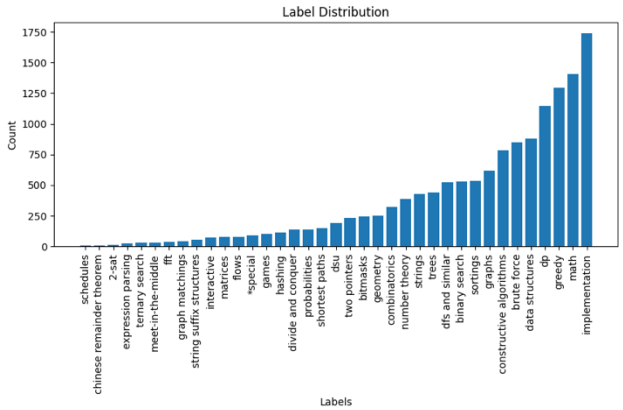
Nhận thấy rằng dữ liệu mã nguồn được thu thập chứa những thông tin dư thừa hoặc thông tin có thể gây tác động không tốt đến việc huấn luyện, chúng tôi quyết định tiền xử lí theo những công đoạn sau:

- Xoá comment, header, include dư thừa của mã nguồn.
- Sử dụng cppcheck để gỡ bỏ những đoạn mã nguồn, biến không được sử dụng.
- Biến đổi tất cả mã nguồn sang chữ thường.
- Xoá những mã nguồn trùng lặp.

Cuối cùng, bộ dữ liệu của chúng tôi có tổng cộng 5550 vấn đề lập trình gồm 36017 mã nguồn.

3.3. Nhãn phân loại

Bởi vì những mã nguồn được lấy từ nền tảng trực tuyến CodeForces, chúng tôi thống nhất sử dụng nhãn được cung cấp bởi CodeForces API. Hơn nữa, chúng tôi đã vẽ đồ thị phân phối nhãn ở **Hình 2** để hiểu thêm về những nhãn, số lượng nhãn mà chúng tôi sử dụng.

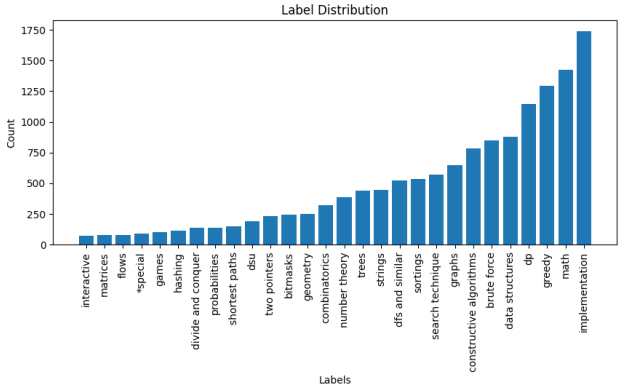


Hình 2: Biểu đồ phân phối nhãn ban đầu

Số lượng nhãn ban đầu của bộ dữ liệu gồm 37 nhãn. Quan sát thấy rằng có một số nhãn chứa rất ít dữ liệu (Schedules, 2-sat,…), chúng tôi quyết định xoá bỏ nhãn hoặc có thể thay thế nhãn bằng những nhãn khác, cụ thể hơn:

- Nhóm nhãn Ternary Search, meet-in-the-middle và Binary Search thay bằng nhãn Search Technique, bởi vì các nhãn đều là các kĩ thuật tìm kiếm.
- Nhóm các kĩ thuật xử lí xâu,cấu trúc dữ liệu xâu hay regular expression thành một nhãn String.

- Nhãn fft và chinese remainder theorem thay thế bằng nhãn math.
- Nhãn graph matching thay thế bằng nhãn graphs.
- Những nhãn có số lượng ít không được đề cập ở trên sẽ bị xoá bỏ.



Hình 3: Biểu đồ phân phối nhãn sau khi xử lí

Số lượng nhãn sau khi xử lí là 28 và chúng tôi sẽ sử dụng số lượng nhãn này là nhãn đầu ra của mô hình của mình. Dễ nhận thấy, bộ dữ liệu của chúng tôi rất mất cân bằng, vì vậy chúng tôi có sử dụng một số kĩ thuật trong việc chia bộ dữ liệu hay hàm mất mát để hạn chế vấn đề này.

3.4. Thống kê

Sau bước tiền xử lí, chúng tôi thống kê mô tả bộ dữ liệu của mình. Cụ thể hơn, chúng tôi sẽ tính toán số lượng vấn đề lập trình, trung bình số mã nguồn và trung bình số nhãn tương ứng với mỗi vấn đề.

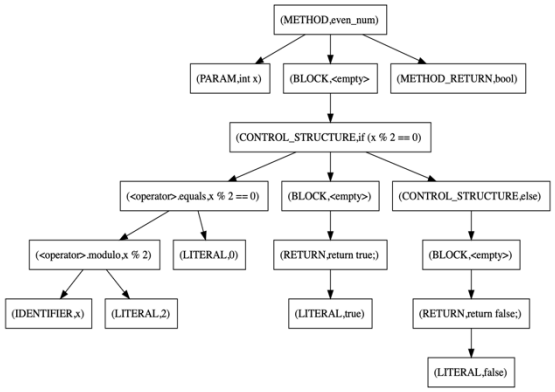
	Vấn đề lập trình	Trung bình số mã nguồn	Trung bình số nhãn
Tổng hợp	5550	6.7183	2.0198

Hình 3: Thống kê bộ dữ liệu

4. Phương pháp

4.1. Biểu diễn mã nguồn theo cấu trúc đường dẫn

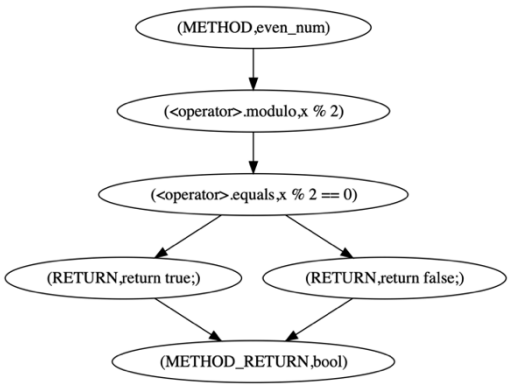
Trong bài báo mocktail [2], họ đã giải thích khái niệm của các đường dẫn biểu diễn AST, CFG và PDG. Sau đây là các khái niệm của chúng:



Hình 4: Cây AST dựa trên mã nguồn hình 1

Định nghĩa 4 (Đường dẫn AST): Gọi n là nút trên cây AST và nó có 2 thuộc tính: loại hình d và tên thể hiện nội dung t. Đường dẫn của cây AST bắt đầu từ nút đầu n_1, đi qua các nút trung gian $n_2,\ldots,n_k$ và kết thúc tại nút cuối n_{k+1} được gọi là đường dẫn AST có độ dài k. Đường dẫn p được biểu diễn dạng chuỗi $d_1a_1d_2a_2\ldots d_ka_kd_{k+1}$, trong đó d_1, d_2, …, d_{k+1} là loại hình của các nút n_1, $n_2,\ldots,n_{k+1}$ và $a_1,a_2,\ldots,a_n$ là hướng di chuyển giữa các nút với $a_i \in \{\uparrow,\downarrow\}$.

Các đường dẫn AST thể hiện cấu trúc của mã nguồn nhưng không nắm bắt được khía cạnh ngữ nghĩa như luồng điều khiển và các phụ thuộc của mã nguồn. Vì vậy, để khắc phục vấn đề này, họ sử dụng thêm 2 đồ thị ngữ nghĩa là CFG và PDG.



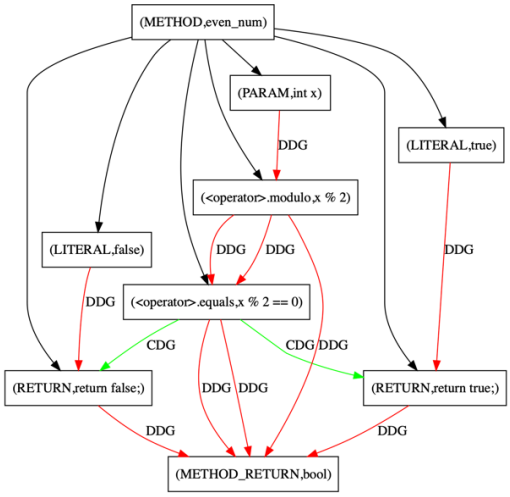
Hình 5: Đồ thị CFG dựa trên mã nguồn hình 1

Định nghĩa 5 (Đường dẫn CFG): Gọi n là nút trên đồ thị CFG và nó có 2 thuộc tính: loại hình d và tên thể hiện nội dung t. Đường dẫn của đồ thị CFG bắt đầu từ nút START n_1, đi qua các nút trung gian $n_{2,\ldots,}n_k$ kết thúc tại nút cuối n_{k+1} được gọi là đường dẫn CFG có độ dài k. Nút kết thúc của CFG có thể là:

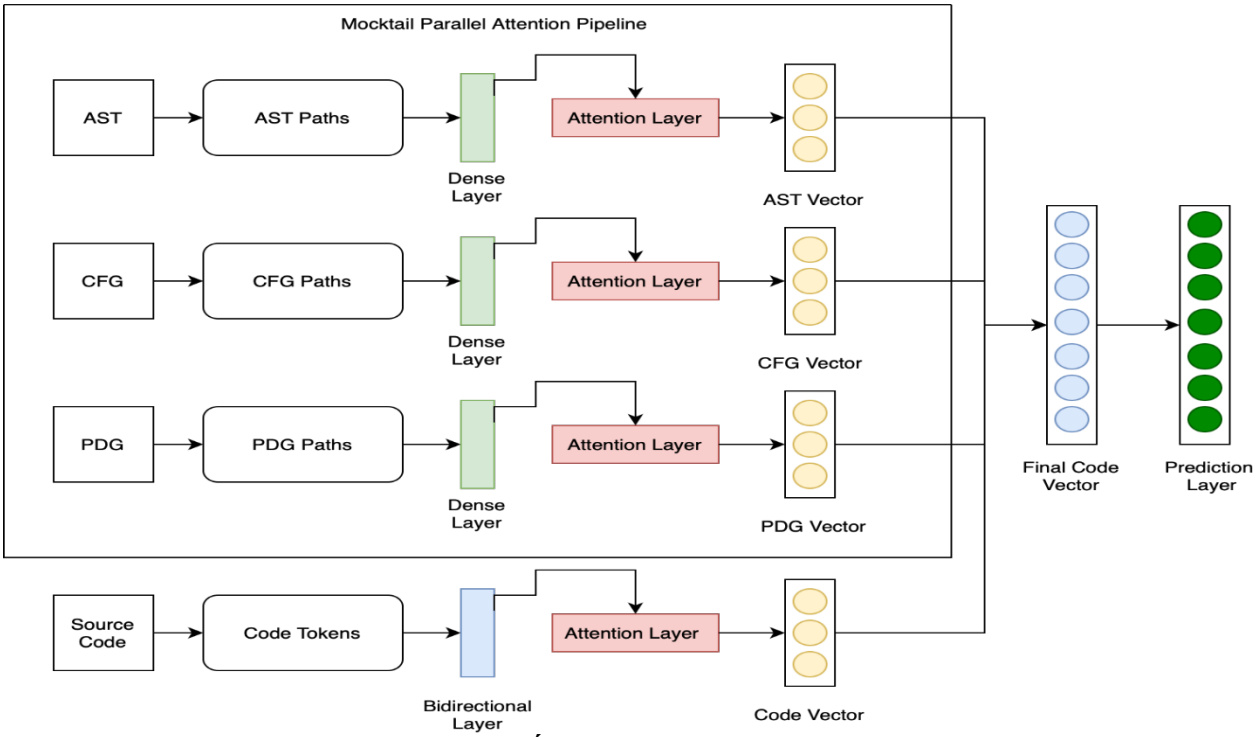
- Nút END
- Nút trung gian đã đi qua thể hiện một vòng lặp.

Đường dẫn p biểu diễn dưới dạng chuỗi $d_1{\downarrow}d_2{\downarrow}\ldots d_ka_kd_{k+1}$, trong đó d_1, d_2, …, d_{k+1} là loại hình của các nút n_1, $n_2,\ldots,n_{k+1}$ và $a_k = \downarrow$ nếu nút kết thúc là nút END, $\uparrow$ nếu ngược lại.

Đường dẫn CFG thể hiện dòng điều khiển trong lúc chương trình chạy.



Hình 6: Đồ thị PDG dựa trên mã nguồn hình 1



Hình 7: Kiến trúc mô hình chú ý song song

Định nghĩa 6 (Đường dẫn PDG): Gọi n là nút trên đồ thị PDG và nó có 2 thuộc tính: loại hình d và tên thể hiện nội dung t. Các cạnh e của đồ thị được gán nhãn l.Một đường dẫn PDG là một chuỗi các nút $n_1,n_2,\ldots,n_{k+1}$ với các cạnh có cùng nhãn l_p. Đường dẫn p của đồ thị được biểu diễn dạng chuỗi $d_1a_1d_2a_2\ldots d_ka_kd_{k+1}$, trong đó $d_1, d_2, \ldots, d_{k+1}$ là loại hình của các nút $n_1, n_2,\ldots,n_{k+1}$ và $a_1,a_2,\ldots,a_n$ là hướng di chuyển giữa các nút. $a_i \in \{\uparrow,\downarrow\}$ và $l_p \in \{CDG, DDG\}$.

Vì PDG là một đồ thị (không phải cây), di chuyển lên và xuống không có ý nghĩa trong PDG. Vì vậy hướng chuyển động giữa các nút đều là ↓, nhằm làm cho tất cả các đường dẫn đều có cùng một định dạng.

4.2. Trích xuất đường dẫn

Để trích xuất đường dẫn chương trình C++, chúng tôi sử dụng mã nguồn mở Joern [3] để trích xuất AST, CFG và PDG. Trong đó, cây AST có độ dài là 8 và độ rộng là 2 như code2vec [1]. Bởi vì số lượng đường dẫn phụ thuộc vào mã nguồn, để tránh vấn đề xuất hiện ma trận thưa cho đầu vào của mô hình, chúng tôi giới hạn đường dẫn bằng cách lấy ngẫu nhiên số lượng AST, CFG và PDG lần lượt là 500, 50, 200. Đồng thời, bởi vì chỉ lấy giới hạn số lượng đường dẫn sẽ dẫn đến việc mất một số thông tin quan trọng của một mã nguồn, chúng tôi quyết định thêm mã nguồn tokens làm đầu vào bằng cách tách bằng mã nguồn mở Tokenizer [9] và cũng giới hạn số lượng là 750.

	AST	CFG	PDG	Source Tokens
Bộ dữ liệu	459.82	32.68	65.68	377.57

Hình 8: Thống kê trung bình đường dẫn

4.3. Mô hình

Trước đó, code2vec [1] đã xây dựng mô hình mạng neuron sử dụng cơ chế chú ý nhưng chỉ với đường dẫn AST. Vì vậy, để kết hợp cả ba đường dẫn AST, CFG và PDG, mocktail đã xây dựng mô hình lấy nhiều loại đầu vào song song. Mocktail [2] lấy đầu vào song song bởi vì AST, CFG và PDG biểu diễn các khía cạnh khác nhau của mã nguồn và cần được học riêng biệt. Vì vậy, các đường dẫn cần có chung một định dạng theo **Định nghĩa 7**.

Định nghĩa 7 (Biểu diễn đường dẫn): Đó là 1 bộ $<t_1, p, t_{k+1}>$, trong đó p là đường dẫn kết nối giữa 2 nội dung của nút.

Các đường dẫn gồm 3 thành phần (1 đường dẫn và 2 nội dung của nút) và cho đi qua 3 lớp Embedding để lấy đặc trưng. Sau đó, 3 lớp Embedding được nối lại và lại cho truyền qua lớp Dense. Ở đây chúng tôi sử dụng hàm kích hoạt Tanh cho lớp Dense. Đầu ra của lớp Dense lại đi qua lớp Attention để tính toán trọng số trung bình đã được giải thích ở phần 2.2. Quá trình này được thực hiện song song với từng đường dẫn và sẽ nhận được các vector trọng số trung bình. Vector mã nguồn cuối cùng là tổng hợp của các vector trọng số trung bình và được sử dụng cho lớp phân loại tuỳ vào mục đích khác nhau. Đây là phân loại đa nhãn vì vậy chúng tôi sử dụng hàm kích hoạt sigmoid cho lớp phân loại.

Để tránh vấn đề thiếu hụt thông tin mã nguồn như đã đề cập ở phần 4.2, đầu vào mã tokens được thêm vào. Các mã tokens này sẽ đi qua một lớp Bidirectional đơn giản và tiếp tục quá trình tính toán vector trọng số trung bình như 3 loại đường dẫn trên.

4.4. Thuật toán Apriori

Mục đích của chúng tôi là phân loại đa nhãn cho một vấn đề lập trình chứ không phải một mã nguồn. Do đó, chúng tôi đã sử dụng thuật toán Apriori [7] để lấy những nhãn xuất hiện nhiều nhất dựa trên đầu ra của mô hình (là tỉ lệ dự đoán nhãn của mỗi mã nguồn). Cho N là số lượng mã nguồn của một vấn đề lập trình. Kết quả của thuật toán

apriori là M tập nhãn C_i với mỗi vấn đề lập trình.

Nếu M = 0, nhãn dự đoán là nhãn có tổng giá trị lớn nhất của N mã nguồn. Nếu M > 0, nhãn dự đoán là giao của M tập nhãn C_i.

5. Triển khai và kết quả thực nghiệm

Trong phần này, chúng tôi cung cấp tổng quan về các thiết lập thực nghiệm. Cụ thể hơn, chúng tôi thảo luận về tập huấn luyện và kiểm tra, hàm đánh giá mà chúng tôi sử dụng, cũng như tham số cho mô hình. Ngoài ra, chúng tôi sẽ trình bày các kết quả của thực nghiệm.

5.1. Tập huấn luyện và kiểm tra

Chúng tôi quyết định thực hiện chia tách Iterative Stratification [10] trên bộ dữ liệu, đồng thời phải dựa theo vấn đề lập trình mà không phải theo mã nguồn với tỉ lệ 80 – 20, trong đó tập huấn luyện gồm 4434 vấn đề với 28895 mã nguồn và tập kiểm tra gồm 1116 vấn đề với 7122 mã nguồn. Quyết định này dựa trên những lí do sau:

- Sử dụng kĩ thuật chia tách Iterative Stratification nhằm mục đích giải quyết vấn đề cân bằng trong bài toán phân loại đa nhãn.
- Bởi vì đang phân loại một vấn đề lập trình, không phải là với một mã nguồn.

5.2. Hàm đánh giá

Bởi vì đây là bài toán phân loại đa nhãn, để đánh giá mô hình của mình, chúng tôi sử dụng hàm đánh giá F1 Score (Micro).

5.3. Tham số mô hình

Dựa vào bảng thống kê **Hình 7**, chiều đầu vào của đường dẫn AST,CFG, PDG và mã tokens lần lượt là 500, 50, 100, 500.

Các lớp Dense, Embedding và Bidirectional có số units là 128.

Chúng tôi huấn luyện mô hình sử dụng thuật toán tối ưu Adam với learning rate là 0.001. Hơn nữa, sử dụng Dropout với giá trị dropout là 0.2 và L2 regularization để hạn chế việc overfitting. Hàm mất mát của mô hình là Focal Loss với gamma = 2, được sử dụng để giải quyết vấn đề mất cân bằng nhãn. Ngưỡng Support của thuật toán Apriori là 0.5. Cuối cùng, lớp phân loại Dense cuối cùng có chiều là 28, ứng với số lượng nhãn cuối cùng sau khi xử lí ở **phần 3.3**.

5.4. Kết quả thực nghiệm

Mô hình	F1 Score (micro)
AST (code2vec)	0.401
CPG (mocktail)	0.426
CPG + Source Code	0.441

Qua kết quả thực nghiệm, đồ thị ngữ nghĩa CFG và PDG giúp mô hình cải thiện hiệu suất so với mô hình chỉ có đầu vào là cây AST. Mô hình CPG + Source Code có F1 Score micro cao nhất là 0.441 so với những mô hình còn lại.

6. Kết luận

Trong nghiên cứu này, chúng tôi đã xây dựng thành công mô hình phân loại đa nhãn các vấn đề lập trình thi đấu với kết quả mà chúng tôi nghĩ là ổn với số lượng nhãn lớn và bộ dữ liệu mất cân bằng. Trong tương lai, việc thu thập dữ liệu là rất quan trọng. Chúng tôi tin rằng mô hình sẽ cho dự đoán tốt hơn với bộ dữ liệu lớn, cân bằng và bao quát hơn và là một bước đệm cho những nghiên cứu sau này trong phân tích ngữ nghĩa của mã nguồn.

Tài liệu tham khảo

[1] Alon, U., Zilberstein, M., Levy, O., & Yahav, E. (2019). code2vec: Learning distributed representations of code. Proceedings of the ACM on Programming Languages, 3(POPL), 1-29.

[2] Vagavolu, D., Swarna, K. C., & Chimalakonda, S. (2021, November). A Mocktail of Source Code Representations. In 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 1296-1300). IEEE.

[3] Yamaguchi, F., Golde, N., Arp, D., & Rieck, K. (2014, May). Modeling and discovering vulnerabilities with code property graphs. In 2014 IEEE Symposium on Security and Privacy(pp. 590-604). IEEE.

[4] Iacob, R. C. A., Monea, V. C., Rădulescu, D., Ceapă, A. F., Rebedea, T., & Trăușan-Matu, Ș. (2020). AlgoLabel: A Large Dataset for Multi-Label Classification of Algorithmic Challenges. Mathematics, 8(11), 1995.

[5] Iancu, B., Mazzola, G., Psarakis, K., & Soilis, P. (2019). Multi-label Classification for Automatic Tag Prediction in the Context of Programming Challenges. arXiv preprint arXiv:1911.12224.

[6] Lin, T. Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal loss for dense object detection. In Proceedings of the IEEE international conference on computer vision (pp. 2980-2988).

[7] Al-Maolegi, M., & Arkok, B. (2014). An improved Apriori algorithm for association rules. arXiv preprint arXiv:1403.3948.

[8] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

[9] A. Mullen, L., Benoit, K., Keyes, O., Selivanov, D., & Arnold, J. (2018). Fast, consistent tokenization of natural language text. *Journal of Open Source Software*, *3*(23), 655.

[10] Sechidis, K., Tsoumakas, G., & Vlahavas, I. (2011, September). On the stratification of multi-label data. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (pp. 145-158). Springer, Berlin, Heidelberg.