Kỹ thuật lựa chọn đặc trưng trong dự đoán code-smell dựa trên học máy

Nguyễn Thanh Bình[1], Nguyễn Hữu Nhật Minh[2], Lê Thị Mỹ Hạnh[3], Nguyễn Thanh Bình[2]
[1]Trường Cao đẳng Cơ điện-Xây dựng và Nông Lâm Trung Bộ, Bình Định
[2]Trường Đại học Công nghệ Thông tin và Truyền thông Việt-Hàn, Đại học Đà Nẵng
[3]Trường Đại học Bách khoa, Đại học Đà Nẵng

Tác giả liên hệ: Nguyễn Thanh Bình, Email: thanhbinh@cdtb.edu.vn
Ngày nhận bài: 15/03/2023, ngày sửa chữa: 20/06/2023, ngày duyệt đăng: 27/06/2023
Định danh DOI: 10.32913/mic-ict-research-vn.v2023.n1.1216

Tóm tắt: Các nghiên cứu thử nghiệm nhằm phát hiện code-smell trong mã nguồn bằng cách sử dụng các kỹ thuật học máy dựa trên tập dữ liệu độ đo mã nguồn đã cho thấy nhiều kết quả hứa hẹn. Các thử nghiệm đã bước đầu đưa các kỹ thuật lựa chọn đặc trưng của các tập dữ liệu vào các mô hình học máy, kết quả thử nghiệm cho thấy các kỹ thuật lựa chọn đặc trưng đã có những tác động tích cực đến hiệu suất dự đoán của các mô hình. Tuy nhiên, chưa có nghiên cứu nào so sánh hiệu suất dự đoán giữa các kỹ thuật lựa chọn đặc trưng trên cùng một mô hình trong dự đoán code-smell. Bài báo này sẽ cung cấp các kết quả thử nghiệm nhằm đánh giá toàn diện các kỹ thuật lựa chọn đặc trưng bằng cách so sánh hiệu suất dự đoán code-smell giữa các kỹ thuật này khi được áp dụng với cùng một mô hình học máy; đồng thời, bài báo này cũng cho biết kỹ thuật lựa chọn đặc trưng phù hợp nhất đối với một mô hình phát hiện code-smell dựa trên học máy.

Từ khóa: *Lựa chọn đặc trưng, code smell và học máy*

Title: **Feature selection techniques in code smell prediction based on machine learning**

Abstract: Recent studies that detect code smells in source code using machine learning techniques based on source code metrics datasets have shown promising results. In this work, we validate feature selection techniques of datasets with machine learning models. As a result, experiments demonstrate that using feature selection techniques could improve predictive performance of the models. To the best of our knowledge, no studies have compared the predictive performance between feature selection techniques on the same model in code smell prediction. In this paper, we would provide the experimental results to comprehensively evaluate feature selection techniques by comparing the code smell predictive performance between these techniques when being applied with the different machine learning models. Throughout extensive evaluation, it shows the best feature selection technique for machine learning-based code smell detection models.

Keywords: *feature selection, code smell, machine learning*

I. GIỚI THIỆU

Trong quá trình bảo trì và phát triển, các hệ thống phần mềm được các lập trình viên cập nhật và phát triển một cách liên tục nhằm mục đích (i) thêm các cài đặt để đáp ứng các yêu cầu mới, (ii) cải thiện các chức năng hiện có, hoặc (iii) sửa các lỗi nghiêm trọng [1]. Do áp lực về thời gian, thiếu kỹ năng hoặc kinh nghiệm, các nhà phát triển không phải lúc nào cũng sẵn sàng kiểm soát hoàn toàn độ phức tạp của hệ thống để tìm ra giải pháp tối ưu trước khi áp dụng các sửa đổi [2]. Do đó, họ thường áp dụng các triển khai dưới mức tối ưu và có khả năng làm sai lệch thiết kế ban đầu của hệ thống.

Code-smell là triệu chứng của việc thiết kế hoặc triển khai dưới mức tối ưu mà các nhà phát triển áp dụng cho hệ thống phần mềm. Do đó, code-smell hiện diện trong mã nguồn có thể dẫn đến phát sinh hoặc có nguy cơ phát sinh lỗi phần mềm trong tương lai. Nhiều nghiên cứu trước đây đã chỉ ra rằng code-smell không chỉ ảnh hưởng đến việc hiểu biết về hệ thống của các nhà phát triển trong các nhiệm vụ bảo trì mà còn ảnh hưởng đến việc thiết kế của hệ thống bị thay đổi và dễ bị lỗi [3–5]. Do đó, việc phát hiện code-smell là một thách thức quan trọng đối với các nhà phát triển để giảm thiểu các nỗ lực và chi phí bảo trì hệ thống [6].

Trong những năm gần đây, các nghiên cứu thử nghiệm nhằm phát hiện code-smell trong mã nguồn bằng cách sử

dụng các kỹ thuật học máy (Machine learning techniques - MLTs) dựa trên tập dữ liệu độ đo mã nguồn (Source code metric dataset - SMD) đã cho thấy nhiều kết quả hứa hẹn [7–13]. Các thử nghiệm này cho thấy SMD là một yếu tố có vai trò quan trọng trong mô hình phát hiện code-smell dựa trên MLTs; tuy nhiên, hầu hết các nghiên cứu đều sử dụng tất cả các đặc trưng hiện hữu của tập dữ liệu (Dataset features - DFs) để huấn luyện cho các mô hình học máy. Đây không phải là một vấn đề quan trọng khi SMD chưa đủ lớn, nhưng cùng với sự phát triển nhanh chóng của các công cụ phân tích mã nguồn hiện nay, dung lượng SMD và số lượng DFs có thể thu được rất lớn, thì nó sẽ là một thách thức đối với chi phí về thời gian và tài nguyên để đáp ứng cho các mô hình phát hiện code-smell dựa trên MLTs.

Bài báo này sẽ thử nghiệm các mô hình phát hiện code-smell hiện diện trong mã nguồn dựa trên các MLTs kết hợp với các kỹ thuật lựa chọn đặc trưng (Feature selection techniques - FSTs) của SMD. FSTs là tập hợp các kỹ thuật nhằm lựa chọn từ SMD các đặc trưng có liên quan với code-smell; với sự hỗ trợ của FSTs, nhiều tập dữ liệu con (SubDataset - SD) sẽ được tạo ra dựa trên các đặc trưng được lựa chọn từ SMD. Các thử nghiệm sẽ được tiến hành bằng cách lần lượt áp dụng các SD cho các MLTs để phát hiện code-smell và trích xuất hiệu suất dự đoán (F1-score) của chúng. Các kết quả thử nghiệm này sẽ được cân nhắc để xác định FST tốt nhất đối với mỗi MLT và code-smell.

Phần tiếp theo của bài báo này là tổng quan về các nghiên cứu liên quan đến vấn đề phát hiện code-smell dựa trên các MLTs. Phần 3 của bài báo sẽ đề cập đến các cân nhắc khi lựa chọn SMD, các kỹ thuật tiền xử lý dữ liệu, các code-smell và FSTs sẽ được áp dụng cho các MLTs của các mô hình thử nghiệm. Ý tưởng thiết kế mô hình thử nghiệm dự đoán code-smell dựa trên học máy kết hợp với các kỹ thuật FSTs được trình bày trong phần 4. Phần 5 sẽ thảo luận liên quan đến kết quả thử nghiệm của các mô hình. Cuối cùng của bài báo sẽ là các kết luận chung liên quan đến các kết quả thử nghiệm và hướng nghiên cứu cần phát triển dựa trên kết quả bài báo này.

II. CÁC NGHIÊN CỨU LIÊN QUAN

Năm 2016, Fontana và các cộng sự [7] đã sử dụng 16 MLTs để dự đoán code-smell trên SMD do nhóm phát triển; Mặc dù các kỹ thuật FSTs không được áp dụng nhưng độ chính xác dự đoán đạt đến 99,10% trong các thử nghiệm của họ. Tiếp tục với tập dữ liệu này, năm 2017 [8], nhóm nghiên cứu tiếp tục công bố kết quả phân loại mức độ nghiêm trọng của code-smell dựa trên các MLTs kết hợp với các kỹ thuật FSTs cơ bản; độ chính xác phân loại của thử nghiệm này đạt đến 93%. Mansoor và các cộng sự [9] đã thử nghiệm dự đoán code-smell trong 5 SMD mà không sử dụng đến các FSTs; độ chính xác của thử nghiệm đạt trung bình 87,00% đối với độ đo *precision* và 92,00% đối với độ đo *recall*.

Một thử nghiệm dự đoán mức độ nghiêm trọng của các lỗi của mã nguồn có sử dụng các kỹ thuật FSTs *Information Gain* và *Chi-square* để chọn các đặc trưng phù hợp từ bộ dữ liệu đã được Pushpalatha và cộng sự thực hiện [10]; độ chính xác dự đoán của mô hình thử nghiệm đạt đến 89,80%. Mohammad Y. Mhawish và các cộng sự đã tiến hành các thử nghiệm dự đoán code-smell dựa trên các mô hình học máy có sử dụng các kỹ thuật FSTs [11, 12] trên tập dữ liệu của Fontana [7]; độ chính xác dự đoán của các thử nghiệm được cải thiện dần từ 98,38% đến 99,70% tùy thuộc vào các kỹ thuật tiền xử lý dữ liệu.

Gần đây, nhóm nghiên cứu của Dewangan đã tiến hành xây dựng mô hình dự đoán code-smell dựa trên 6 MLTs [13] để dự đoán code-smell *Long-method* trên tập dữ liệu của Fontana [7]; họ đã áp dụng các kỹ thuật FSTs *Chi-square* và *Wrapper based* để lựa chọn các đặc trưng phù hợp với mô hình thử nghiệm, đồng thời phương pháp xác thực *10-Fold- Cross-Validation* được áp dụng để xác thực độ tin cậy của việc dự đoán code-smell của mô hình thử nghiệm. Kết quả thử nghiệm cho thấy dự đoán code-smell *Feature-envy* đạt độ chính xác đến 99,12%.

Các nghiên cứu trên đã bước đầu đưa các kỹ thuật FSTs của tập dữ liệu vào các mô hình để tăng hiệu suất dự đoán. Kết quả thử nghiệm cho thấy các kỹ thuật FSTs đã có những tác động tích cực đến hiệu suất dự đoán của các mô hình. Tuy nhiên, chưa có nghiên cứu nào so sánh khả năng tăng hiệu suất giữa các kỹ thuật FSTs đối với cùng một mô hình dự đoán code-smell. Vì vậy, bài báo này sẽ bổ sung cho các các nghiên cứu trên trong hai vấn đề chính:

(i) Cung cấp một nghiên cứu nhằm đánh giá toàn diện các FSTs bằng cách so sánh hiệu suất dự đoán code-smell giữa các FSTs khi được áp dụng với cùng một mô hình thử nghiệm.

(ii) Đồng thời, bài báo này cũng cho biết kỹ thuật FST tốt nhất đối với một mô hình phát hiện code-smell dựa trên MLTs.

III. LỰA CHỌN CÁC FSTS

Việc trích xuất độ đo từ mã nguồn thường tạo ra các SMD có số lượng các đặc trưng khá lớn tùy thuộc vào công cụ được sử dụng; Mặc dù vậy, không phải lúc nào các các đặc trưng hiện diện trong SMD đều liên quan trực tiếp đến các code-smell cần được phát hiện, cho nên các MLTs có thể tạo ra kết quả kém chính xác và khó hiểu hơn hoặc có thể không phát hiện ra bất kỳ điều gì hữu ích.

FSTs là các phương pháp cố gắng xác định và loại bỏ càng nhiều các đặc trưng không liên quan và dư thừa càng tốt trước khi áp dụng vào các MLTs. FSTs có thể nâng cao

hiệu suất, tiết kiệm thời gian nhờ việc giảm không gian tìm kiếm trên các biến độc lập và trong một số trường hợp, giảm yêu cầu lưu trữ. Hiện nay, các phương pháp lựa chọn đặc trưng được chia làm ba nhóm sau: phương pháp Filter, phương pháp Wrapper và phương pháp Embedded.

1. Phương pháp Filter (Filter methods)

a) Pearson correlation (F-Corr):

Pearson correlation là một hệ số có giá trị từ -1 đến 1; giá trị này cho biết mức độ mà hai đặc trưng có liên quan tuyến tính với nhau. Nếu Pearson correlation = 0, nghĩa là chúng không có liên quan với nhau. Mục đích của kỹ thuật này là xác định các đặc trưng có hệ số Pearson correlation cao và loại bỏ chúng khỏi tập đặc trưng của SMD.

b) Analysis of Variance (F-Aov):

là một kỹ thuật thống kê kiểm tra xem các đặc trưng đầu vào khác nhau có giá trị khác nhau đáng kể cho biến đầu ra hay không. Kỹ thuật cho phép phân tích nhiều nhóm dữ liệu để xác định sự khác biệt giữa các mẫu và trong các mẫu, để có được thông tin về mối quan hệ giữa các biến phụ thuộc và biến độc lập giúp chúng ta lựa chọn các đặc trưng tốt nhất.

c) Mutual information (F-MI):

MI là lượng thông tin có thể thu được từ một biến ngẫu nhiên cho một biến khác. Giá trị này thể hiện mức độ phụ thuộc lẫn nhau giữa các biến độc lập và biến phụ thuộc. MI luôn lớn hơn hoặc bằng 0, trong đó giá trị càng lớn thì mối quan hệ giữa hai biến càng lớn. Nếu MI= 0, thì các biến là độc lập.

2. Phương pháp Wrapper (Wrapper methods)

a) Forward selection (W-Fws):

Một đặc trưng tốt nhất được chọn để bắt đầu và thêm nhiều lần lặp lại. Ở mỗi lần lặp lại tiếp theo, các đặc trưng tốt nhất còn lại được thêm vào dựa trên tiêu chí hiệu suất của chúng.

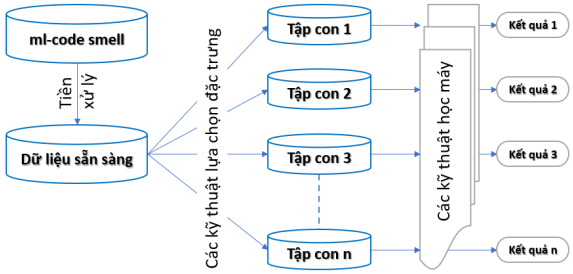
b) Backward elimination (W-Bwe):

Kỹ thuật này bắt đầu với tất cả các đặc trưng và lặp đi lặp lại loại bỏ từng đặc trưng một. Một trong những thuật toán phổ biến nhất là Recursive Feature Elimination (RFE) giúp loại bỏ các đặc trưng ít quan trọng hơn dựa trên xếp hạng mức độ quan trọng của chúng.

3. Phương pháp Embedded (Embedded methods)

Các phương pháp thuộc nhóm Embedded kết hợp các ưu điểm của cả hai phương pháp Filter và phương pháp Wrapper. Các phương pháp này hiệu quả hơn các phương pháp Wrapper và chính xác hơn các phương pháp Filter; trong các phương pháp Embedded, việc lựa chọn đặc trưng được thực hiện trong quá trình huấn luyện. Hai phương pháp lựa chọn đặc trưng thuộc nhóm Embedded là ***Hồi quy Ridge*** **(E-Ridge)** và ***Hồi quy Lasso*** **(E-Lasso)** được lựa chọn để đưa vào các mô hình thử nghiệm.

IV. THIẾT KẾ VÀ THỬ NGHIỆM MÔ HÌNH DỰ ĐOÁN CODE-SMELL KẾT HỢP FSTS



Hình 1. Mô hình thử nghiệm xác định hiệu suất của các kỹ thuật lựa chọn đặc trưng

1. Lựa chọn SCM và kỹ thuật tiền xử lý dữ liệu

Bài báo sử dụng tập dữ liệu **ml-Codesmell** của nhóm tác giả và đã được công bố [14]. Tập dữ liệu này đã được đề xuất cho các mô hình dự đoán code-smell dựa trên phương pháp học máy với một lượng lớn quan sát cùng với nhiều đặc trưng được tạo bằng cách thu thập các độ đo mã nguồn được trích xuất từ 525 dự án nguồn mở.

Tập dữ liệu ml-Code-smell được phân tách thành 2 tập dữ liệu con: tập dữ liệu huấn luyện *(training)* để huấn luyện các MLTs và tập dữ liệu kiểm tra *(testing)* để kiểm tra hiệu suất dự đoán của các MLTs đối với sự hiện diện của các code-smell có trong tập này. Dựa trên kết quả đã thu được qua thử nghiệm các tỷ lệ phân tách *training:testing* khác nhau trên cùng 1 tập dữ liệu và kỹ thuật học máy [14], tỷ lệ phân tách dữ liệu *training:testing* là *80:20* được chọn để áp dụng cho các thử nghiệm của bài báo này.

Kỹ thuật cân bằng dữ liệu SMOTE được áp dụng để xử lý mất cân bằng dữ liệu trong tập dữ liệu huấn luyện. SMOTE là một trong các kỹ thuật lấy mẫu quá mức *(Oversampling)*; trong đó các mẫu tổng hợp được tạo ra cho lớp thiểu số. Thuật toán này giúp khắc phục vấn đề *over-fitting* do quá trình lấy mẫu ngẫu nhiên *(Random Oversampling)* gây ra. Nó tập trung vào không gian các đặc trưng để tạo ra các mẫu quan sát mới với sự trợ giúp của phép nội suy giữa lớp thiểu số nằm cùng nhau.

2. Lựa chọn các code-smell cho mô hình thử nghiệm

Tập dữ liệu ml-Codesmell [1], bao gồm 8 code-smell ở cấp độ lớp *(Class-level)* và 6 code-smell ở cấp độ phương thức *(Method-level)*, được thể hiện trong Bảng I.

Bảng I
TÓM LƯỢC THÔNG TIN VỀ TẬP DỮ LIỆU ML-CODESMELL

Cấp độ code-smell	Dung lượng	Số lượng mẫu	Số lượng đặc trưng	Số lượng code-smell
Class-level	76 MB	373,400	41	8
Method-level	491 MB	2,486,282	33	6

Trong phạm vi bài báo, để tập trung vào mục tiêu đánh giá hiệu suất của các FSTs và tiết kiệm thời gian, năm code-smell ở cấp độ lớp *(Brain-Class, Data-Class, God-Class, Model-Class, và Schizophrenic-Class)* được lựa chọn để đưa vào mô hình thử nghiệm.

3. Các mô hình học máy (MTLs)

Các mô hình học máy được tạo ra với nhiều thuật toán khác nhau. Năm thuật toán được chọn dựa trên mức độ phổ biến, tính đơn giản và hiệu suất cao của chúng. Các thuật toán này được mô tả ngắn gọn như sau:

- **Random Forest Classifier (RFC)**: RFC phù hợp với một số bộ phân loại cây quyết định dựa vào các tập con khác nhau của SMD và tập hợp các cây đầu ra theo giá trị trung bình của từng cây. Độ chính xác đầu ra của RFC phụ thuộc vào sức mạnh của từng cây trong rừng và mối tương quan giữa chúng. Do đó, RFC có thể cải thiện độ chính xác dự đoán và tránh vấn đề khớp quá mức (over-fitting) [15].
- **Light Gradient Boosting (LGB)**: LGB xây dựng một mô hình dựa trên các cây quyết định đơn giản không được tối ưu hóa tốt và sau đó khái quát hóa chúng bằng cách tối ưu hóa một hàm mất mát (loss-function) được xác định tùy ý để đưa ra các dự đoán mạnh mẽ hơn. Các ưu điểm của LGB như sau: (i) có tốc độ huấn luyện nhanh hơn và hiệu quả cao hơn nhiều thuật toán khác nhưng tiêu tốn bộ nhớ thấp hơn, (ii) có độ chính xác cao hơn bất kỳ thuật toán tăng tốc nào khác và (iii) có khả năng tương thích tốt hơn với các bộ dữ liệu lớn [16].
- **K-nearest Neighbors Classifier (KNN)**: Các quan sát được phân loại dựa trên lớp của mẫu lân cận gần nhất của chúng. KNN có hai giai đoạn: (i) Xác định các mẫu gần nhất và (ii) xác định lớp sử dụng các lân cận đó. Một số ưu điểm của KNN như sau: (i) nhanh chóng triển khai và gỡ lỗi, (ii) rất hiệu quả nếu các lân cận được sử dụng phân tích để làm giải thích và (iii) có nhiều kỹ thuật có thể nhanh chóng tăng độ chính xác và cải thiện đáng kể thời gian quá trình thử nghiệm trên các tập dữ liệu lớn [17].
- **Linear Logistic Regression (LLR)**: Thuật toán mạnh mẽ này cho phép nhiều biến độc lập được phân tích đồng thời và giảm các tác động gây nhiễu bằng cách phân tích mối liên hệ của tất cả các biến. LLR phân loại các mẫu dựa trên xác suất của một sự kiện, vì vậy các biến phụ thuộc và độc lập của hồi quy logistic không cần mối tương quan [18, 19].
- **Linear Support Vector Classification (SVM)**: SVM là một mô hình học máy có giám sát (supervised learning model) với sự kết hợp các thuật toán học máy nhằm phân tích dữ liệu để phân loại và phân tích hồi quy. SVM hoạt động tương đối tốt khi có biên độ phân cách rõ ràng giữa các lớp. SVM hiệu quả hơn trong không gian nhiều chiều và tương đối hiệu quả về bộ nhớ [20].

4. Lựa chọn các độ đo hiệu suất để đánh giá mô hình

Sau khi được huấn luyện với *training*, các MLTs có thể dự đoán một mẫu trong *testing* là *dương tính*(là code-smell) hay là *âm tính* (không phải code-smell). Kết quả dự đoán của mô hình có thể rơi vào một trong 4 trường hợp sau:

- TP (True-Positive): Dự đoán đúng 1 mẫu *dương tính*.
- TN (True-Negative): Dự đoán đúng 1 mẫu *âm tính*.
- FP (False-Positive): Dự đoán sai 1 mẫu *dương tính*.
- FN (False-Negative): Dự đoán sai 1 mẫu *âm tính*.

Sau khi xác định được các giá trị trên, độ chính xác dự đoán của mô hình thử nghiệm được đánh giá bằng các độ đo cơ bản sau:

- **Accuracy** *Accuracy* là một độ đo dùng để đánh giá hiệu suất dự đoán của một MLTs. Giá trị *Accuracy* càng cao thì độ chính xác dự đoán càng lớn. *Accuracy* được tính toán dựa trên công thức sau:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

Tuy nhiên, khi tập dữ liệu bị mất cân bằng nghiêm trọng, việc sử dụng *Accuracy* làm độ đo đánh giá hiệu suất của mô hình thường không hiệu quả vì hầu hết chúng đều có độ chính xác rất cao; bất kỳ mô hình nào có thể dự đoán tất cả các mẫu thuộc về lớp đa số cũng sẽ cho kết quả gần 100%. Do đó, các độ đo như *Precision*, *Recall* và *F1-score* thường được cân nhắc để thay thế cho *Accuracy*. Các độ đo này đề cao tính chính xác của lớp thiểu số hơn là lớp đa số.

[1] https://doi.org/10.6084/m9.figshare.21343299.v2

- **Precision** Độ đo này nhằm cố gắng trả lời câu hỏi sau: Tỷ lệ nhận dạng đúng các mẫu *dương tính* là bao nhiêu? *Precision* được định nghĩa như sau:

$$Precision = \frac{TP}{TP + FP} \quad (2)$$

- **Recall** *Recall* định lượng tỷ lệ dự đoán đúng mẫu *dương tính* từ tất cả các dự đoán *dương tính* đã được thực hiện. Không giống như *Precision* chỉ trả về các dự đoán chính xác mẫu *dương tính* trong số tất cả các dự đoán *dương tính*, *Recall* còn cung cấp dấu hiệu của các dự đoán *dương tính* bị bỏ lỡ. Công thức tính *Recall* như sau:

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

- **F1-score** Đây là một độ đo hài hòa giữa *Precision* và *Recall*, đồng thời cũng là độ đo lý tưởng để đánh giá các mô hình có bộ dữ liệu mất cân bằng nghiêm trọng.

$$F1\text{-}score = \frac{2(Precision * Recall)}{Precision + Recall} \quad (4)$$

Trong bài báo này, *F1-score* được sử dụng làm thước đo hiệu suất của các mô hình dự đoán dựa trên MLTs.

V. KẾT QUẢ VÀ THẢO LUẬN

1. Kết quả dự đoán đối với các code-smell

a) Brain-Class:

Kết quả dự đoán code-smell Brain-Class được thể hiện trong Bảng II. Hầu hết các FSTs đều cho hiệu suất tốt hơn so với SMD nguyên bản, ngoại trừ đối với mô hình MLTs áp dụng giải thuật LGB; Nhìn chung, hiệu suất dự đoán của các MLTs đối với code-smell này không cao; tuy nhiên, khi LGB sử dụng FSTs là F-Corr thì hiệu suất đạt đến 95%, tương đương với SMD nguyên bản.

Bảng II
KẾT QUẢ HIỆU SUẤT DỰ ĐOÁN CODE-SMELL BRAIN-CLASS

FSTs \ MLTs	KNN	LGB	LRR	RFC	SVM
None-FST	0,30	0,95	0,15	0,47	0,43
E-Lasso	0,31	0,66	0,12	0,48	0,46
E-Ridge	0,26	0,64	**0,32**	0,5	0,42
F-Aov	0,33	0,58	0,30	0,50	0,48
F-Corr	0,33	**0,95**	0,17	**0,5**	0,45
F-MI	0,28	0,63	0,24	0,48	**0,51**
W-Bwe	**0,5**	0,68	0,24	0,39	0,5
W-Fws	0,42	0,55	0,21	0,40	0,20

b) Data-Class:

Số liệu của kết quả dự đoán được thể hiện trong Bảng III. Hiệu suất dự đoán của các MLTs đối với code-smell Data-Class vượt trội và ổn định so với Brain-Class; Hầu hết các FSTs đều có hiệu suất dự đoán đạt từ 99% trở lên tùy theo MLTs được chọn.

Bảng III
KẾT QUẢ HIỆU SUẤT DỰ ĐOÁN CODE-SMELL DATA-CLASS

FSTs \ MLTs	KNN	LGB	LRR	RFC	SVM
None-FST	0,85	1	0,95	1	1
E-Lasso	0,86	**1**	0,9	**1**	**1**
E-Ridge	0,96	0,99	0,92	0,97	0,95
F-Aov	0,98	1	0,95	0,99	0,96
F-Corr	0,89	1	**0,96**	0,99	0,99
F-MI	0,99	1	0,95	0,99	0,95
W-Bwe	**0,99**	1	0,95	0,96	0,99
W-Fws	0,98	1	0,95	0,99	1

c) God-Class:

Kết quả hiệu suất thử nghiệm với code-smell God-Class, được trình bày trong Bảng IV, cho thấy chỉ có 2 MLTs là LGB và RFC là phù hợp với code-smell này khi cho thấy hiệu suất dự đoán đạt từ 99%; tuy nhiên, các kỹ thuật FSTs không phát huy hiệu quả khi hiệu suất của các mô hình không tỏ ra vượt trội so với SMD nguyên bản.

Bảng IV
KẾT QUẢ HIỆU SUẤT DỰ ĐOÁN CODE-SMELL GOD-CLASS

FSTs \ MLTs	KNN	LGB	LRR	RFC	SVM
None-FST	0,62	**1**	0,36	0,97	0,53
E-Lasso	0,64	0,96	0,24	0,95	0,48
E-Ridge	0,6	0,96	0,5	0,95	0,47
F-Aov	0,62	0,96	**0,54**	0,95	0,3
F-Corr	0,56	0,97	0,24	0,87	0,59
F-MI	**0,71**	1	0,44	**0,99**	0,58
W-Bwe	0,65	0,87	0,52	0,66	**0,64**
W-Fws	0,65	0,87	0,52	0,66	0,64

d) Model-Class:

Hầu hết các MLTs được đưa vào mô hình tỏ ra phù hợp với code-smell Model-Class bởi vì chúng đều trả về hiệu suất rất cao (từ 98% trở lên) khi thử nghiệm, ngoại trừ kỹ thuật SVM; tuy nhiên, bên cạnh đó, các phương pháp FSTs lại không có hiệu quả như kỳ vọng khi hiệu suất của chúng chênh lệch không đáng kể so với SMD nguyên bản mặc dù tiêu tốn khá nhiều thời gian cho quá trình loại bỏ bớt các DFs không liên quan và dư thừa (kết quả thử nghiệm có trong Bảng V).

Bảng V
KẾT QUẢ HIỆU SUẤT DỰ ĐOÁN CODE-SMELL MODEL-CLASS

FSTs \ MLTs	KNN	LGB	LRR	RFC	SVM
None-FST	1	1	0,98	1	0,77
E-Lasso	1	1	**0,98**	1	0,64
E-Ridge	1	1	0,98	1	**1**
F-Aov	0,98	0,98	0,96	0,97	0,92
F-Corr	1	**1**	0,98	**1**	0,24
F-MI	**1**	1	0,98	1	0,31
W-Bwe	0,97	0,98	0,94	0,98	0,91
W-Fws	0,97	0,98	0,94	0,98	0,91

e) Schizophrenic-Class

Khi thử nghiệm dự đoán code-smell Schizophrenic-Class bằng các mô hình áp dụng các MLTs LGB và RFC, các FSTs không cho thấy sự chênh lệch hiệu suất dự đoán so với SMD nguyên bản bởi vì tất cả chúng đều trả về hiệu suất 100% (Bảng VI); tuy nhiên, khi dùng các mô hình được áp dụng các MLTs còn lại, các FSTs đã thể hiện được tính hiệu quả của chúng: tăng hiệu suất từ 58% lên 86% đối với MLTs KNN, tăng từ 84% lên 99% đối với MLTs LRR và tăng từ 40% lên 53% đối với MLTs SVM.

Bảng VI
KẾT QUẢ HIỆU SUẤT DỰ ĐOÁN CODE-SMELL SCHIZOPHRENIC-CLASS

FSTs \ MLTs	KNN	LGB	LRR	RFC	SVM
None-FST	0,58	1	0,84	1	0,4
E-Lasso	0,64	**1**	0,69	1	**0,53**
E-Ridge	0,73	1	0,88	1	0,34
F-Aov	0,66	1	0,98	1	0,4
F-Corr	0,63	1	0,84	**1**	0,35
F-MI	0,75	1	**0,99**	1	0,4
W-Bwe	**0,86**	1	0,99	1	0,34
W-Fws	0,79	1	0,99	1	0,4

2. Thảo luận chung về các kết quả thử nghiệm

Kết quả thử nghiệm cho thấy sau khi tập dữ liệu được áp dụng các kỹ thuật FSTs, hiệu suất dự đoán của các mô hình MLTs đã có những cải thiện tích cực tùy theo từng FSTs và MLTs. Dựa vào các kết quả thử nghiệm trên, các FSTs đã thể hiện một số đặc điểm tích cực sau:

- Cải thiện độ chính xác của mô hình: Bằng cách chọn kỹ thuật FSTs phù hợp với MLTs, việc lựa chọn đặc trưng đã thể hiện khả năng cải thiện độ chính xác của mô hình thử nghiệm. Tất cả các thử nghiệm đều cho thấy với mỗi MLTs đều có ít nhất một kỹ thuật FSTs cho hiệu suất bằng hoặc cao hơn với tập dữ liệu nguyên bản SMD.

- Cải thiện khả năng diễn giải: Bằng cách chỉ chọn các đặc trưng phù hợp nhất, các kỹ thuật FSTs có thể cải thiện khả năng diễn giải của mô hình, giúp dễ hiểu và giải thích các đặc trưng góp phần vào dự đoán của mô hình.

- Giảm dung lượng tập dữ liệu: Các FSTs giúp giảm kích thước của tập dữ liệu, điều này có thể giúp trực quan hóa và phân tích dữ liệu dễ dàng hơn.

- Giảm thời gian huấn luyện các mô hình: Bằng cách giảm số lượng đặc trưng, FSTs có thể giảm độ phức tạp tính toán của mô hình, vì vậy thời gian huấn luyện các mô hình được giảm đi đáng kể.

Nhìn chung, kết quả thử nghiệm đã cho thấy các FSTs giúp giảm thiểu thời gian huấn luyện, cải thiện độ chính xác, hiệu quả và khả năng diễn giải của các mô hình học máy, giúp chúng trở nên hiệu quả và hữu ích hơn cho nhiều ứng dụng. Tuy nhiên, quá trình thử nghiệm cũng cho thấy một số vấn đề cần cân nhắc khi áp dụng các FSTs:

- Tốn thời gian: FSTs có thể là một quá trình tốn nhiều thời gian, đặc biệt khi làm việc với các bộ dữ liệu lớn hoặc sử dụng các kỹ thuật thuộc nhóm phương pháp Wrapper.

- Mất thông tin: Việc xóa các đặc trưng khỏi tập dữ liệu có thể dẫn đến mất thông tin có giá trị tiềm ẩn, điều này có thể ảnh hưởng đến độ chính xác của mô hình. Kết quả thử nghiệm cho thấy nhiều FSTs đã cho hiệu suất dự đoán thấp hơn so với tập dữ liệu nguyên bản (None-FSTs).

Vì vậy, khi áp dụng các FSTs, điều quan trọng là phải đánh giá cẩn thận các ưu điểm và nhược điểm tiềm ẩn của từng kỹ thuật FSTs và chọn kỹ thuật tốt nhất dựa trên các mục tiêu và yêu cầu cụ thể của mô hình học máy.

VI. KẾT LUẬN

Đến phần cuối bài báo này, qua kết quả thử nghiệm, có thể kết luận rằng FSTs là các kỹ thuật về việc giữ các đặc trưng hữu ích và có liên quan một cách tự động thông qua các quy trình đã được áp dụng trong các mô hình thử nghiệm. Bài báo đã thử nghiệm các kỹ thuật FSTs được xây dựng từ các quan điểm khác nhau: từ số liệu thống kê đến phương pháp học máy. Nhìn chung, khi áp dụng FSTs cho các MLTs, có thể dẫn đến các mô hình tốt hơn, đạt được hiệu suất cao hơn và dễ hiểu hơn. Cuối cùng nhưng không kém phần quan trọng, đó là việc có một tập hợp con các đặc trưng cho phép mô hình học máy được huấn luyện nhanh hơn mà vẫn cải thiện được hiệu suất dự đoán. Về hướng nghiên cứu sâu hơn đối với các kỹ thuật FSTs, việc kết hợp nhiều kỹ thuật FSTs để lựa chọn các đặc trưng cần được tiếp cận; một tập hợp các kỹ thuật FSTs phù hợp có thể cải thiện được hiệu suất và giảm thiểu thời gian huấn luyện cho các mô hình học máy trên tập dữ liệu siêu lớn.

TÀI LIỆU THAM KHẢO

[1] M. Lehman, “Programs, life cycles, and laws of software evolution,” *Proceedings of the IEEE*, vol. 68, pp. 1060–1076, 1980.

[2] D. Parnas, “Software aging.” IEEE Comput. Soc. Press, 1994, pp. 279–287.

[3] P. Avgeriou and P. Kruchten, “Managing technical debt in software engineering,” *Dagstuhl Seminar 16162*, 2016.

[4] F. Khomh, M. D. Penta, Y.-G. Guéhéneuc, and G. Antoniol, “An exploratory study of the impact of antipatterns on class change- and fault-proneness,” *Empirical Software Engineering*, vol. 17, pp. 243–275, 8 2012.

[5] D. Spadini, F. Palomba, A. Zaidman, M. Bruntink, and A. Bacchelli, “On the relation of test smells to software code quality,” *2018 IEEE International*

Conference on Software Maintenance and Evolution (ICSME), pp. 1–12, 8 2018.

[6] A. Yamashita and L. Moonen, "Do code smells reflect important maintainability aspects?" *IEEE International Conference on Software Maintenance, ICSM*, pp. 306–315, 8 2012.

[7] F. A. Fontana, M. V. Mäntylä, M. Zanoni, and A. Marino, "Comparing and experimenting machine learning techniques for code smell detection," *Empirical Software Engineering*, vol. 21, pp. 1143–1191, 6 2016.

[8] F. A. Fontana and M. Zanoni, "Code smell severity classification using machine learning techniques," *Knowledge-Based Systems*, vol. 128, pp. 43–58, 7 2017.

[9] U. Mansoor, M. Kessentini, B. R. Maxim, and K. Deb, "Multi-objective code-smells detection using good and bad design examples," *Software Quality Journal*, vol. 25, pp. 529–552, 6 2017.

[10] P. M. N and M. M, "Predicting the severity of open source bug reports using unsupervised and supervised techniques," *International Journal of Open Source Software and Processes*, vol. 10, pp. 1–15, 1 2019.

[11] M. Y. Mhawish and M. Gupta, "Generating code-smell prediction rules using decision tree algorithm and software metrics," *International Journal of Computer Sciences and Engineering*, vol. 7, pp. 41–48, 5 2019.

[12] ——, "Predicting code smells and analysis of predictions: Using machine learning techniques and software metrics," *Journal of Computer Science and Technology*, vol. 35, pp. 1428–1445, 11 2020.

[13] S. Dewangan and R. S. Rao, *Code Smell Detection Using Classification Approaches*, 2022.

[14] B. N. Thanh, M. N. N. H, H. L. T. My, and B. N. Thanh, "Ml-codesmell: A code smell prediction dataset for machine learning approaches." Association for Computing Machinery, 12 2022, pp. 368–374.

[15] L. Breiman, "Random forests," pp. 5–32, 2001.

[16] D. A. McCarty, H. W. Kim, and H. K. Lee, "Evaluation of light gradient boosted machine learning technique in large scale land use and land cover classification," *Environments*, vol. 7, p. 84, 10 2020.

[17] P. Cunningham and S. J. Delany, "k-nearest neighbour classifiers - a tutorial," *ACM Computing Surveys*, vol. 54, pp. 1–25, 7 2022.

[18] S. Sperandei, "Understanding logistic regression analysis," *Biochemia Medica*, vol. 24, pp. 12–18, 2014.

[19] C.-Y. J. Peng, K. L. Lee, and G. M. Ingersoll, "An introduction to logistic regression analysis and reporting," *The Journal of Educational Research*, vol. 96, pp. 3–14, 9 2002.

[20] G. Fung and O. L. Mangasarian, "Semi-supervised support vector machines for unlabeled data classification," pp. 1–14, 2001.

Nguyễn Thanh Bình tốt nghiệp Thạc sỹ chuyên ngành Khoa học máy tính năm 2011 và đang làm Nghiên cứu sinh chuyên ngành Khoa học máy tính tại Đại học Đà Nẵng. Hiện công tác tại Phòng Đào tạo và Hợp tác quốc tế, Trường Cao đẳng Cơ điện-Xây dựng và Nông Lâm Trung Bộ, Bình Định.
Email: thanhbinh@cdtb.edu.vn

Nguyễn Hữu Nhật Minh tốt nghiệp Đại học chuyên ngành Khoa học và Kỹ thuật Máy tính, Trường Đại học Bách Khoa TP. Hồ Chí Minh năm 2013. Năm 2020 bảo vệ Tiến sĩ ngành Khoa học và Kỹ thuật Máy tính của trường Đại học Kyung Hee, Hàn Quốc. Hiện là giảng viên, nghiên cứu viên của trường Đại học Công nghệ Thông tin & Truyền thông Việt Hàn, Đại học Đà Nẵng., phụ trách Chương trình nghiên cứu Viện Khoa học và Công nghệ Số của trường. Các lĩnh vực nghiên cứu chính bao gồm wireless communications, mobile edge computing, federated learning, xử lý ngôn ngữ tự nhiên và xử lý hình ảnh.
Email: nhnminh@vku.udn.vn

Lê Thị Mỹ Hạnh hiện là giảng viên Khoa Công nghệ Thông tin, Trường Đại học Bách Khoa Đà Nẵng, Việt Nam. Nhận bằng Thạc sỹ năm 2004 và bằng Tiến sĩ Khoa học Máy tính năm 2016, tại Đại học Đà Nẵng. Các lĩnh vực nghiên cứu gồm Công nghệ phần mềm và Ứng dụng các kỹ thuật heuristic cho các vấn đề trong công nghệ phần mềm.
Email: ltmhanh@dut.udn.vn

Nguyễn Thanh Bình nhận bằng Tiến sĩ Công nghệ Thông tin tại Trường Đại học Bách Khoa Grenoble, Cộng hòa Pháp năm 2004 và đã được công nhận Phó Giáo sư từ năm 2013. Hiện công tác tại Trường Đại học Công nghệ Thông tin và Truyền thông Việt-Hàn, Đại học Đà Nẵng. Các lĩnh vực nghiên cứu chính bao gồm Công nghệ phần mềm và Chất lượng phần mềm.
Email: ntbinh@vku.udn.vn