DOI: 10.15625/vap.2020.00237

XỬ LÝ DỮ LIỆU KHÔNG CÂN BẰNG TRONG BÀI TOÁN DỰ ĐOÁN LỖI PHẦN MỀM

Lê Song Toàn[1], Nguyễn Thanh Bình[2], Lê Thị Mỹ Hạnh[3], Nguyễn Thanh Bình[1]

[1]Trường Đại học Công nghệ Thông tin và Truyền thông Việt - Hàn, Đại học Đà Nẵng

[2]Trường Cao đẳng Cơ điện - Xây dựng và Nông Lâm Trung Bộ

[3]Khoa Công nghệ thông tin, Trường Đại học Bách khoa, Đại học Đà Nẵng

lstoan@vku.udn.vn, thanhbinh@cdntrungbo.edu.vn,ltmhanh@dut.udn.vn, ntbinh@vku.udn.vn

TÓM TẮT: *Dự đoán lỗi phần mềm giúp dự đoán trước khả năng có lỗi của mã nguồn, làm giảm thời gian kiểm thử, tăng chất lượng của sản phẩm. Các đặc trưng mã nguồn là những thông tin quan trọng giúp việc dự đoán lỗi phần mềm chính xác. Tuy nhiên, nhiều bộ dữ liệu về dự báo lỗi bị mất cân bằng, tức là số lượng dữ liệu giữa các lớp có sự chênh lệch lớn. Trong bài báo này, chúng tôi nghiên cứu về mất cân bằng dữ liệu và sự cần thiết của việc lấy mẫu dữ liệu để xử lý dữ liệu không cân bằng. Chúng tôi tiến hành thử nghiệm với ba kỹ thuật lấy mẫu dữ liệu để xử lý dữ liệu không cân bằng, gồm: Random Undersampling, Random Oversampling và SMOTE. Các kỹ thuật được áp dụng vào ba tập dữ liệu về dự báo lỗi của NASA trong kho lưu trữ của Promise. Các kết quả thu được cho thấy tính hiệu quả của việc áp dụng các mô hình học máy kết hợp với các kỹ thuật lấy mẫu để nâng cao tính chính xác của mô hình dự đoán lỗi phần mềm.*

Từ khóa: *Fault prediction, Undersampling, Oversampling, SMOTE.*

I. ĐẶT VẤN ĐỀ

Lĩnh vực phần mềm hiện đang được áp dụng ngày càng rộng rãi và phổ biến trong hầu như tất cả các lĩnh vực của cuộc sống, chúng được sử dụng để tự động hóa và vận hành hầu hết các quy trình nghiệp vụ mà chúng ta có hiện nay. Thách thức với công nghệ phần mềm hiện đại là các hệ thống phần mềm ngày càng trở nên phức tạp và tinh vi hơn. Vì vậy, việc đảm bảo chất lượng phần mềm càng khó khăn hơn. Trong đó, xác định lỗi cũng như dự báo lỗi phần mềm đóng vai trò quan trọng.

Dự đoán chính xác lỗi trong các dự án phần mềm nhằm giảm thiểu các rủi ro về chất lượng và nâng cao khả năng phát hiện các thành phần chương trình có tồn tại lỗi hay không. Do đó, các phương pháp phân loại lỗi để dự đoán khả năng có lỗi dựa trên các đặc trưng tĩnh về mã nguồn đã trở thành một chủ đề nhận được nhiều sự quan tâm của cộng đồng nghiên cứu trong những năm gần đây.

Các mô hình dự đoán lỗi phần mềm được xem là một trong những giải pháp hữu ích và tiết kiệm chi phí nhất để xác định lỗi có thể xảy ra. Trong số các mô hình về dự đoán lỗi, phương pháp sử dụng các độ đo về kích thước và độ phức tạp của mã nguồn là khá phổ biến. Các mô hình này thực hiện trên các thành phần mã nguồn chương trình làm cơ sở để xác định các thành phần nào có nguy cơ xảy ra lỗi hoặc dự đoán được số lỗi trong từng đoạn mã nguồn.

Tính chất không cân bằng về số mẫu dữ liệu trong mỗi lớp đã trở thành sự cản trở lớn đến hiệu quả của các phương pháp học kết hợp nhiều thuật toán. Để giải quyết vấn đề này, nhiều phương pháp đã được đề xuất để giải quyết vấn đề này và được phân thành hai nhóm cơ bản: tiếp cận ở mức giải thuật và tiếp cận ở mức dữ liệu. Các phương pháp tiếp cận ở mức giải thuật hướng tới việc điều chỉnh các thuật toán phân lớp để có hiệu quả cao trên các tập dữ liệu mất cân bằng. Nghiên cứu này chỉ tập trung vào phương pháp tiếp cận ở mức dữ liệu nhằm nhắm tới thay đổi sự phân bố các đối tượng bằng cách sinh thêm các phần tử cho lớp thiểu số, hay giảm bớt các phần tử thuộc lớp đa số để làm giảm sự mất cân bằng giữa các lớp đối tượng.

Trong các phần tiếp theo, chúng tôi sẽ trình bày chi tiết quá trình thiết lập thực nghiệm và đánh giá hiệu quả của việc có hay không nên sử dụng kỹ thuật lấy mẫu đối với một tập dữ liệu trong bài toán dự đoán lỗi phần mềm. Phần 2 sẽ trình bày tổng quan về dữ liệu không cân bằng trong bài toán dự đoán lỗi phần mềm; Phần 3 trình bày chi tiết về các kỹ thuật lấy mẫu dữ liệu được thực hiện trong nghiên cứu này; Phần 4 là mô hình xử lý dữ liệu không cân bằng; Phần 5 trình bày cụ thể việc thiết lập thực nghiệm và kết quả thực nghiệm, từ đó chỉ ra tính hiệu quả của việc áp dụng các kỹ thuật để nâng cao tính chính xác của kết quả dự đoán lỗi phần mềm.

II. DỰ ĐOÁN LỖI PHẦN MỀM

2.1. Dự đoán lỗi phần mềm

Dự đoán lỗi trong phần mềm (SDP – Software Defect Prediction) [1] là quá trình xác định các thành phần của hệ thống phần mềm có thể chứa lỗi hay không. Các mô hình SDP được xây dựng bởi ba cách tiếp cận:

- Sử dụng các thuộc tính có thể đo được của hệ thống phần mềm được gọi là độ đo mã nguồn;
- Sử dụng dữ liệu lỗi từ một dự án phần mềm tương tự;
- Dựa vào các phiên bản phần mềm đã phát hành trước đó.

Sau khi được xây dựng, mô hình SDP có thể được áp dụng cho các dự án phần mềm trong tương lai và do đó có thể xác định các thành phần dễ bị lỗi của hệ thống phần mềm.

Việc phát triển các mô hình dự đoán lỗi phần mềm được xem là một trong những giải pháp được ưu tiên hàng đầu. Các mô hình thường sử dụng các độ đo về kích thước và độ phức tạp được phân tích từ các dòng mã lệnh trong chương trình để xử lý. Các mô hình dự đoán này giả định rằng lỗi được xảy ra do kích thước và độ phức tạp của chương trình gây ra.

Từ các công trình trước đây [2], các nhà nghiên cứu đã chỉ ra rằng kỹ thuật học máy là cách tiếp cận tốt nhất để dự đoán mức độ lỗi. Các mô hình dự đoán lỗi phần mềm thường sử dụng bộ dữ liệu lỗi của NASA có sẵn từ kho lưu trữ của Promise và áp dụng phương pháp xác thực chéo. Theo cách tiếp cận chéo k, tập dữ liệu ban đầu được chia ngẫu nhiên thành k tập: D1, D2,..., Dk; các tập có kích thước xấp xỉ bằng nhau. Kỹ thuật được huấn luyện và thử nghiệm với k lần. Trong mỗi lần, một tập được sử dụng để kiểm tra và các tập còn lại được sử dụng để huấn luyện mô hình. Quá trình được lặp lại cho đến khi mỗi lần k được sử dụng làm tập kiểm tra. Độ chính xác tổng thể được ước tính bằng cách lấy trung bình các kết quả của mỗi vòng lặp. Mô hình dự đoán phụ thuộc vào chất lượng của các bộ dữ liệu được sử dụng, do đó việc chọn bộ dữ liệu đóng vai trò quan trọng.

2.2. Độ đo mã nguồn

Độ đo phần mềm được xem là độ đo định lượng gán các ký hiệu hoặc số cho các đặc điểm của các thành phần được dự đoán. Chúng mô tả nhiều thuộc tính như độ tin cậy, độ phức tạp và chất lượng của các sản phẩm phần mềm. Những độ đo này đóng một vai trò quan trọng trong việc xây dựng một bộ dự báo lỗi phần mềm hiệu quả.

Độ đo mã nguồn được thu thập trực tiếp từ mã nguồn của phần mềm. Những độ đo này đánh giá độ phức tạp của mã nguồn dựa trên giả định rằng các thành phần phần mềm phức tạp có nhiều khả năng chứa lỗi hơn. Có nhiều độ đo mã nguồn khác nhau đã được sử dụng để dự đoán lỗi phần mềm. Để dự đoán số lỗi, Akiyama sử dụng số dòng mã lệnh làm độ đo duy nhất [3]. Tuy nhiên, chỉ sử dụng độ đo này quá đơn giản để đo độ phức tạp của sản phẩm phần mềm.

Vì vậy, các độ đo khác [4] thường được sử dụng trong các mô hình dự đoán lỗi. Các độ đo này được gọi là các thuộc tính mã tĩnh được giới thiệu bởi McCabe [5] và Halstead [6]. Các thuộc tính Halstead được chọn lựa trên độ phức tạp của mã nguồn. Trong khi, các thuộc tính của McCabe là các độ đo chu trình thể thiện sự phức tạp của một sản phẩm phần mềm. Khác với các thuộc tính Halstead, các thuộc tính của McCabe đo độ phức tạp của cấu trúc mã nguồn. Chúng thu được bằng cách tính số lượng các thành phần, các cung và nút được kết nối trong các biểu đồ luồng điều khiển của mã nguồn. Mỗi nút của biểu đồ luồng biểu diễn một câu lệnh tuần tự chương trình, trong khi một cung biểu diễn lệnh rẽ nhánh.

Các độ đo của McCabe, Halstead và của Akiyama là những độ đo tiêu biểu của các độ đo hướng phương thức.

2.3. Tập dữ liệu không cân bằng

Trong bài toán dự đoán lỗi, dữ liệu thu được trong các bộ dữ liệu phân tích từ các mã nguồn chương trình thường là các tập dữ liệu mất cân bằng. Số lượng mẫu có lỗi chỉ chiếm thiểu số so với số mẫu không lỗi, điều này dẫn đến việc các mô hình dự đoán lỗi có thể tính toán và phân tích hệ thống phần mềm sai lệch, có thể gây ra những lỗi ngoài ý muốn.

Các thuật toán học phân loại tiêu chuẩn thường nghiêng về lớp đa số và do đó có tỷ lệ phân loại sai cao hơn cho các trường hợp của lớp thiểu số. Do đó, các thuật toán học máy và các kỹ thuật lấy mẫu dữ liệu được cải tiến hoặc được nghiên cứu mới nhằm giải quyết vấn đề này.

Mặc dù một số mô hình đã cho thấy có hiệu quả trong các nghiên cứu thực nghiệm, các mô hình như vậy đáp ứng tốt trong việc dự đoán lớp đa số nhưng lại kém hơn trong dự đoán lớp thiểu số (các môđun bị lỗi). Vấn đề này làm nảy sinh yêu cầu cần phải xây dựng một tập dữ liệu huấn luyện có tính cân bằng giữa các loại mẫu.

Tuy nhiên, Menzies et *al*. đã chỉ ra rằng các bộ dữ liệu khác nhau dẫn đến độ chính xác dự đoán khác nhau [8] và thường độ chính xác phụ thuộc vào độ đo hoặc tính năng khác nhau thay đổi từ bộ dữ liệu này sang bộ dữ liệu khác. Ngoài ra, trong trường hợp xảy ra sự cố dự đoán lỗi, bộ dữ liệu thường có vấn đề mất cân bằng lớp, tức là, số lượng các trường hợp đại diện cho lớp "bị lỗi" ít hơn nhiều so với số lượng các trường hợp đại diện cho lớp "không bị lỗi".

Do đó, các kỹ thuật lấy mẫu khác nhau thường được sử dụng để giải quyết vấn đề này. Lấy mẫu dữ liệu được sử dụng để chọn, xử lý và phân tích một tập hợp con đại diện của các điểm dữ liệu để xác định các mẫu và xu hướng trong tập dữ liệu lớn hơn đang được kiểm tra. Nó cho phép việc lập mô hình dự đoán và phân tích dữ liệu làm việc với một lượng dữ liệu nhỏ, có thể quản lý được, để xây dựng và thực hiện các mô hình phân tích nhanh hơn, trong khi vẫn tạo ra kết quả có độ chính xác cao.

III. XỬ LÝ DỮ LIỆU KHÔNG CÂN BẰNG VÀ CÁC KỸ THUẬT LẤY MẪU DỮ LIỆU

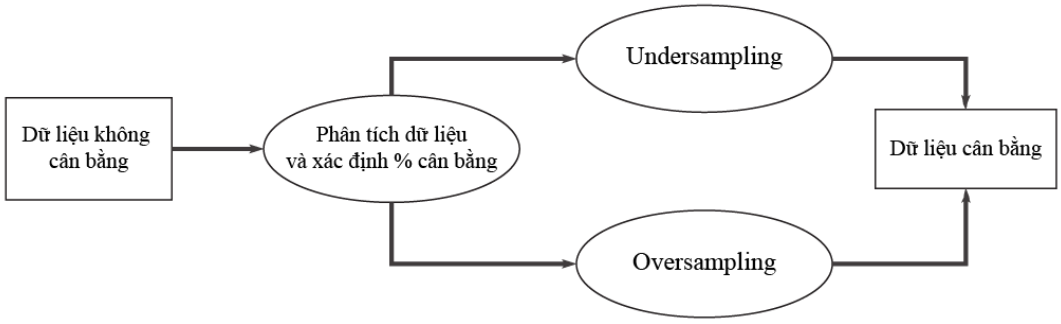
Trong phạm vi bài báo này, chúng tôi sẽ tập trung nghiên cứu việc xử lý dữ liệu không cân bằng bởi phương pháp sử dụng các kỹ thuật lấy mẫu dữ liệu.

3.1. Xử lý dữ liệu không cân bằng

Trong bài toán dự đoán lỗi thì số lượng mẫu có lỗi chỉ chiếm thiểu số so với số mẫu không lỗi, điều này dẫn đến việc các mô hình dự đoán lỗi có thể tính toán và phân tích hệ thống phần mềm sai lệch, có thể gây ra những lỗi ngoài ý

muốn. Việc dự đoán đúng nhãn của mẫu thuộc lớp nhỏ là cần thiết và quan trọng. Nếu mẫu thuộc lớp nhỏ có nhãn sai thì phần mềm sẽ phát sinh lỗi bất kỳ lúc nào và nhà phát triển phần mềm phải trả chi phí và nhân lực cho việc phân tích, chi phí sửa lỗi cao hơn nhiều so với việc gán đúng nhãn.

Lấy mẫu dữ liệu đặc biệt hữu ích với các tập dữ liệu quá lớn để phân tích đầy đủ một cách hiệu quả. Việc xử lý dữ liệu không cân bằng được mô hình hoá bởi Hình 1.



Hình 1: Mô hình xử lý dữ liệu không cân bằng

Với mỗi tập dữ liệu, mô hình xử lý sẽ tính toán số lượng mẫu lỗi/không lỗi và tỷ lệ mất cân bằng giữa 2 lớp. Dựa vào số lượng mẫu của mỗi lớp đã tính toán đó, tùy theo mỗi kỹ thuật, tính toán số lượng mẫu tăng (đối với kỹ thuật Oversampling) hoặc giảm (đối với kỹ thuật Undersampling) hoặc lai tạo (đối với kỹ thuật SMOTE) số mẫu để tập dữ liệu được cân bằng. Tiến hành áp dụng chiến lược lấy mẫu của mỗi kỹ thuật lên từng tập dữ liệu.

Việc mất cân bằng dữ liệu ảnh hưởng xấu đến những mô hình học máy trong vấn đề giải quyết bài toán dự đoán lỗi phần mềm. Những mô hình này khi được học những tập dữ liệu mất cân bằng, sẽ cho kết quả dự đoán không như mong muốn. Những phần mềm, ứng dụng được mô hình dự đoán sẽ cho ra kết quả không tốt và nhà phát triển phần mềm sẽ tiêu tốn thêm thời gian và chi phí để giải quyết vấn đề do chính mô hình gây ra.

Nhìn chung, các kỹ thuật lấy mẫu dữ liệu được sử dụng để giải quyết bài toán phân bố mẫu giữa các lớp không cân bằng có thể được chia thành hai nhóm tăng mẫu và giảm mẫu. Các phương pháp tăng mẫu (Oversampling) bổ sung thêm các mẫu cho các lớp có số dữ liệu ít, trong khi các kỹ thuật giảm mẫu (Undersampling) loại bỏ bớt các mẫu của các lớp có số lượng mẫu nhiều với mục đích thu được tập dữ liệu cân bằng hơn.

3.2. Kỹ thuật Undersampling

Undersampling là một nhóm các kỹ thuật được thiết kế để cân bằng lớp cho một tập dữ liệu có phân phối lớp bị lệch. Các kỹ thuật lấy mẫu loại bỏ các mẫu dữ liệu khỏi tập dữ liệu ở lớp đa số để cân bằng tốt hơn với lớp thiểu số, chẳng hạn như giảm độ lệch từ 1:100 xuống 1:10, 1:2 hoặc thậm chí là 1:1. Bảng 1 trình bày một số các kỹ thuật thuộc nhóm Undersampling.

Bảng 1. Các kỹ thuật Undersampling

Random Undersampling	Kỹ thuật lấy mẫu đơn giản nhất bao gồm chọn ngẫu nhiên các mẫu dữ liệu từ lớp đa số và xóa chúng khỏi tập dữ liệu huấn luyện. Kỹ thuật này loại bỏ mẫu dữ liệu một cách ngẫu nhiên.
Cluster	Cluster là một phương pháp làm cho hầu hết các nhóm hoặc cụm trong mẫu đại diện cho tổng số các mẫu liên quan đến đặc tính mà chúng ta muốn đo lường. Chúng ta có thể chọn chỉ một vài trong số các cụm này để tiến hành nghiên cứu.
Tomek links	Tomek links loại bỏ sự chồng chéo không mong muốn giữa các lớp, các liên kết lớp đa số được loại bỏ cho đến khi tất cả các cặp lân cận gần nhất được phân tách tối thiểu là cùng một lớp. Ý tưởng là làm rõ ranh giới giữa các nhóm thiểu số và đa số, làm cho (các) khu vực thiểu số trở nên khác biệt hơn.

Trong phạm vi nghiên cứu này, đối với các kỹ thuật Undersampling, chúng tôi chọn kỹ thuật Random Undersampling để xử lý dữ liệu không cân bằng trong bài toán dự đoán lỗi phần mềm. Random Undersampling là một cách tiếp cận lấy mẫu rất đơn giản, loại bỏ ngẫu nhiên một số mẫu ở lớp đa số để cân bằng tập dữ liệu [9].

Chiến lược của Random Undersampling:

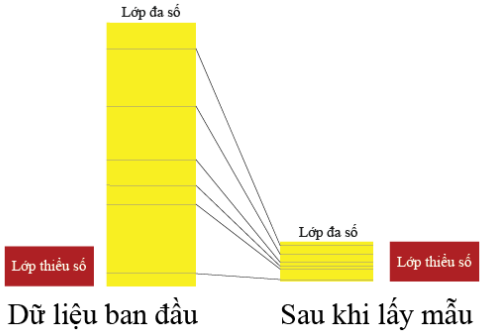
1. Phân tích dữ liệu và xác định lớp thiểu số và đa số;
2. Tính toán số lượng phiên bản cần loại bỏ dựa trên tỷ lệ phần trăm của mẫu;
3. Xác định một mẫu ngẫu nhiên từ lớp đa số và loại bỏ nó khỏi lớp đa số;
4. Lặp lại bước 3 cho đến khi số lượng phiên bản được loại bỏ theo tỷ lệ phần trăm nhất định.

Dựa vào đó, cách tiếp cận này là cân bằng tập dữ liệu bằng cách loại bỏ ngẫu nhiên các mẫu khỏi các tập dữ liệu. Trong trường hợp này, số lượng mẫu được loại bỏ sẽ được tự động tính toán để đảm bảo rằng tổng số mẫu lỗi và

không lỗi được cân bằng. Thuật toán cũng cho phép đặc tả tỷ lệ phần của kỹ thuật thông qua tham số u. Tỷ lệ phần trăm u <1 xác định số lượng mẫu được lấy trong mỗi tập.

Kỹ thuật Random Undersampling thực hiện giữ tất cả các mẫu trong lớp thiểu số và chọn ngẫu nhiên một số lượng mẫu bằng với lớp thiểu số trong lớp đa số để tạo ra tập dữ liệu mới được cân bằng.

Mặc dù phương pháp đơn giản, tốn ít chi phí về thời gian cũng như bộ nhớ cho quá trình phân lớp, nhưng hạn chế của kỹ thuật này là các mẫu được loại bỏ mà không cần quan tâm đến mức độ hữu ích hoặc quan trọng của chúng. Điều này có nghĩa là thông tin hữu ích đó sẽ bị xóa.



Hình 2. Mô tả thuật toán Undersampling

3.3. Kỹ thuật Oversampling

Ngược lại với Undersampling, các kỹ thuật Oversampling bổ sung thêm mẫu ở lớp thiểu số để đưa tập dữ liệu đạt mức cân bằng. Oversampling làm tăng số lượng mẫu của lớp thiểu số bằng cách sao chép các mẫu của lớp thiểu số. Mặc dù nó không làm tăng thông tin mẫu. Nhưng với một số kỹ thuật khác nhau, nó làm tăng thêm các trường hợp khác nhau ở tập mẫu mới (Bảng 2 trình bày một số kỹ thuật tăng mẫu), điều này khiến cho tập dữ liệu quá cụ thể. Nó làm tăng độ chính xác cho mô hình học máy, nhưng hiệu suất cho các bộ dữ liệu mới thực sự kém hơn vì tập dữ liệu mới có thể xuất hiện các trường hợp không có thực.

Bảng 2. Các kỹ thuật Oversampling

Random Oversampling	Random Oversampling bổ sung dữ liệu với nhiều bản sao của một số mẫu được chọn ngẫu nhiên ở lớp thiểu số. Thay vì sao chép mọi mẫu trong lớp thiểu số, một số trong số chúng có thể được chọn ngẫu nhiên với sự thay thế.
SMOTE	SMOTE sử dụng các mẫu dữ liệu được tổng hợp. Dữ liệu mới này được tạo bằng cách nội suy giữa một số trường hợp ở lớp thiểu số nằm trong vùng lân cận được xác định
ADASYN	ADASYN được phát triển trên phương pháp SMOTE, bằng cách chuyển tầm quan trọng sang các lớp thiểu số khó học. Dữ liệu tổng hợp được tạo ra nhiều hơn cho các mẫu lớp thiểu số khó học. ADASYN không chỉ có thể làm giảm xu hướng học được đưa ra ban đầu bởi phân lớp dữ liệu mất cân bằng, mà còn có thể thay đổi thích ứng ranh giới quyết định để tập trung vào những mẫu khó học.

Trong các phần tiếp theo, chúng tôi sẽ trình bày chi tiết kỹ thuật là Random Oversampling và kỹ thuật SMOTE.

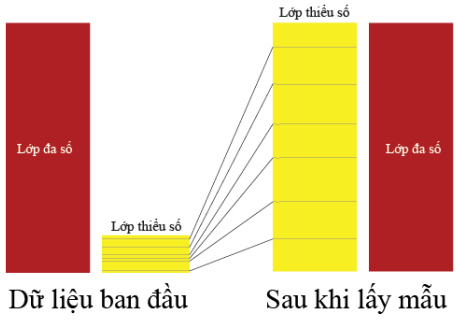
A. Kỹ thuật Random Oversampling

Kỹ thuật Random Oversampling [9] là phương pháp sinh thêm mẫu ngẫu nhiên ở lớp thiểu số (mẫu lỗi) để cân bằng tỷ lệ. Chiến thuật của Random Oversampling được thực hiện bằng cách:

- Giữ nguyên số lượng mẫu không lỗi ở lớp đa số;
- Tăng số lượng mẫu được chọn ngẫu nhiên ở lớp thiểu số;

Trong chiến lược này, mục tiêu là cân bằng số lượng các mẫu lỗi và không lỗi với việc chọn ngẫu nhiên một mẫu từ lớp thiểu số, sau đó tạo các bản sao của mẫu bằng cách nhân mẫu đó với một hệ số để gia tăng số lượng mẫu, tiếp tục thực hiện việc tạo bản sao của mẫu lỗi khác đến khi tập dữ liệu được cân bằng. Tham số tùy chọn *o* cho phép chọn một tỷ lệ phần trăm cụ thể để áp dụng trong tập dữ liệu với các giá trị liên quan.

Kỹ thuật này có nhược điểm là số lượng mẫu lỗi được tăng lên gấp nhiều lần nhưng nội dung và chất lượng của mẫu không có sự thay đổi. Trong trường hợp tập dữ liệu có kích thước lớn thì chi phí thời gian và bộ nhớ cho giai đoạn phân lớp sẽ gia tăng đáng kể.



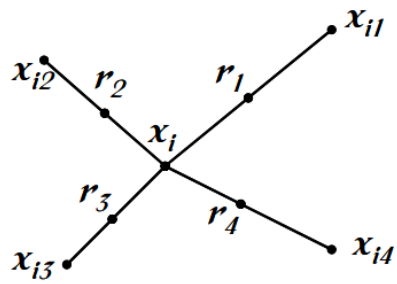
Hình 3. Mô tả dữ liệu với thuật toán Oversampling

B. SMOTE

Chawla và các cộng sự [10] đã đề xuất kỹ thuật SMOTE (Synthetic Minority Over-sampling Technique) nhằm cải tiến kỹ thuật Random Oversampling. Ý tưởng là khắc phục những nhược điểm của các kỹ thuật Oversampling hay thậm chí là kỹ thuật Undersampling bằng cách sao chép thuộc tính mẫu dữ liệu và hỗ trợ trình phân loại để cải thiện tính khái quát của nó trên dữ liệu thực nghiệm. Cơ sở của kỹ thuật SMOTE là tạo ra các trường hợp thiểu số mới. SMOTE tiến hành kiểm tra tất cả các thuộc tính của từng mẫu dữ liệu trong lớp thiểu số và tăng số lượng mẫu của nhóm thiểu số bằng cách đưa ra các mẫu về nhóm dữ liệu ở lớp thiểu số mới.

SMOTE thực hiện cách tiếp cận như Oversampling để cân bằng lại tập huấn luyện ban đầu. Thay vì áp dụng như Random Oversampling, ý tưởng của SMOTE là giới thiệu các mẫu dữ liệu được tổng hợp [10]. Dữ liệu mới này được tạo bằng cách nội suy giữa một số trường hợp ở lớp thiểu số nằm trong vùng lân cận được xác định. Vì lý do này, kỹ thuật SMOTE tập trung vào tính năng của dữ liệu, nói cách khác, thuật toán dựa trên các giá trị của các tính năng và mối quan hệ của chúng, thay vì coi các mẫu dữ liệu là ưu tiên. Điều này cũng đã dẫn đến việc nghiên cứu mối quan hệ giữa các trường hợp ban đầu và tổng hợp phải được phân tích sâu, bao gồm cả chiều sâu dữ liệu. Một mẫu dữ liệu của lớp thiểu số x_i được chọn làm cơ sở để tạo các điểm dữ liệu tổng hợp mới. Dựa trên một thước đo khoảng cách, một số điểm gần nhất của cùng một lớp (điểm x_{i1} đến x_{i4}) được chọn từ tập huấn luyện. Cuối cùng, một phép nội suy ngẫu nhiên được thực hiện để thu được các mẫu mới r_1 đến r_4 [10].

Cách SMOTE hoạt động như sau: Đầu tiên, tổng số lượng bội số N (một giá trị số nguyên) được thiết lập, có thể được thiết lập để có được sự phân phối lớp là 1:1 gần đúng. Sau đó, một quá trình lặp đi lặp lại được thực hiện, bao gồm một số bước. Một mẫu của lớp thiểu số được chọn ngẫu nhiên từ tập huấn luyện. Tiếp theo, K lân cận của nó được lấy.



Hình 4. Mô tả thuật toán của kỹ thuật SMOTE

Cuối cùng, N của các mẫu K này được chọn ngẫu nhiên để tính toán để trở thành mẫu mới bằng phép nội suy. Để làm như vậy, sự khác biệt giữa vectơ đặc trưng (mẫu) đang được xem xét và từng nút lân cận được chọn sẽ được lấy. Sự khác biệt này được nhân với một tham số số ngẫu nhiên trong khoảng từ 0 đến 1 và sau đó nó được thêm vào vectơ tính năng trước đó.

IV. THIẾT LẬP THỰC NGHIỆM VÀ ĐÁNH GIÁ KẾT QUẢ

4.1. Các độ đo đánh giá thực nghiệm

Trong phạm vi nghiên cứu này, chúng tôi sẽ sử dụng độ đo F1-score và đường cong ROC (Receiver Operating Characteristic) tính được từ các thuật toán huấn luyện trên các tập dữ liệu không cân bằng trước và sau khi được lấy mẫu. Dựa vào đó có thể tính toán để đánh giá tính hiệu quả của mỗi kỹ thuật lấy mẫu dữ liệu. Các kết quả thu được cho thấy tính hiệu quả của việc áp dụng các mô hình học kết hợp với các kỹ thuật lấy mẫu để nâng cao tính chính xác của kết quả dự đoán lỗi phần mềm.

Để đánh giá hiệu quả của các thuật toán phân lớp, người ta thường sử dụng độ đo Accuracy [11], được tính bằng tỉ lệ giữa số kết quả dự đoán đúng và số kết quả dự đoán được. Mỗi bài toán phân lớp nhị phân có một ma trận nhầm lẫn:

Bảng 3. Ma trận nhầm lẫn

	Thực tế có lỗi	Thực tế không lỗi
Dự đoán có lỗi	TP Dương tính thật (*True Positive*)	FP Dương tính giả (*False Positive*)
Dự đoán không lỗi	FN Âm tính giả (*False Negative*)	TN Âm tính thật (*True Negative*)

Đối với bài toán dự đoán lỗi phần mềm sẽ có 4 khả năng dự đoán có thể xảy ra, nếu một mẫu được phân lớp là có lỗi và thực sự nó là lỗi thì đó là dương tính thật, nếu một mẫu không lỗi bị phân loại nhầm thành có lỗi thì nó là dương tính giả. Tương tự như vậy, âm tính thật chỉ ra trường hợp mẫu được dự đoán là không lỗi và thực sự nó cũng không có lỗi, trong khi âm tính giả chỉ ra một trường hợp dự đoán một mẫu có lỗi thành không lỗi. Dương tính giả dẫn đến lãng phí nguồn lực dành cho việc giám sát một thành phần chương trình đã có chất lượng tốt, ngược lại âm tính giả gây ra lỗi nghiêm trọng hơn vì nó dẫn đến việc bỏ lỡ cơ hội tìm thấy một thành phần bị lỗi thực sự.

Độ chính xác Accuracy của một thuật toán phân lớp trên một tập dữ liệu đã cho được định nghĩa như công thức.

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

Ngoài ra, các phép đo đánh giá khác thường được sử dụng trong cộng đồng nghiên cứu để tạo ra các đánh giá toàn diện hơn cho bài toán học từ dữ liệu không cân bằng.

Precision: Số đoạn mã có lỗi được phân loại đúng trên tổng số đoạn mã được phân loại có lỗi. Tiêu chí này tượng trưng cho độ chính xác phân loại lỗi của mô hình. Precision càng lớn thì mô hình dự đoán lỗi càng chính xác [12]:

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

Recall: Số đoạn mã có lỗi được phân loại đúng trên tổng số đoạn mã có lỗi. Tiêu chí này tượng trưng cho độ nhạy với lỗi của thuật toán [12].

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

F1-score là trung bình giữa Precision và Recall [12]:

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

Ngoài 3 độ đo trên, ở các bài toán phân lớp nhị phân, đường cong ROC còn được sử dụng để đánh giá hiệu quả của các mô hình dự đoán. Đường cong càng đi dọc về phía biên trái và biên trên, diện tích dưới đường cong AUC (Area under the Curve) càng lớn thì mô hình dự đoán càng hiệu quả, ngược lại, đường cong càng tiến đến đường chéo 45^o, AUC càng bé thì mô hình càng kém hiệu quả.

4.2. Các bộ dữ liệu thực nghiệm

Để đánh giá hiệu quả của các kỹ thuật lấy mẫu trong bài toán dự đoán lỗi phần mềm, chúng tôi đã tiến hành các thực nghiệm trên 03 tập dữ liệu hướng phương thức (CLASS, MC1, PC1) từ các dự án của NASA có số mẫu và số thuộc tính khác nhau cũng như tỉ lệ không cân bằng về số mẫu trong các lớp cũng khác nhau. Bảng 4 tóm tắt các thuộc tính của mỗi tập dữ liệu không cân bằng được sử dụng gồm tổng số thuộc tính, số mẫu, số mẫu được gán nhãn lỗi, số mẫu được gán nhãn không lỗi, tỉ lệ số mẫu lỗi trên tổng số mẫu.

Ba tập dữ liệu này sử dụng các phép đo mức thiết kế và các phép đo mức mã nguồn. Phép đo mức thiết kế bao gồm số nút, số nhánh, độ phức tạp thiết kế và phép đo độ phức tạp chu trình McCabe. Các phép đo mã nguồn tĩnh là các đặc trưng được trích xuất ra từ mã nguồn. Một số phép đo mã nguồn tĩnh gồm có số toán tử, số toán hạng, phép đo Halstead được tính toán từ các câu lệnh. Các tính chất của mỗi tập dữ liệu bao gồm 20 phép đo mã nguồn đo độ phức tạp, tính phụ thuộc, tính gắn kết, kích thước và các tính chất dễ gây ra lỗi trong một lớp của hệ thống phần mềm.

Như đã trình bày ở Mục III, các tập dữ liệu thực nghiệm sẽ được xử lý qua các bước như mô hình xử lý đã thiết lập. Tổng số tập đầu vào mô hình xử lý là 03 tập dữ liệu lấy từ các dự án của NASA, vậy tổng số tập đầu ra là 09 tập dữ liệu đã cân bằng tương ứng với 3 kỹ thuật lấy mẫu.

Thực nghiệm tiếp tục sử dụng 12 tập dữ liệu trên làm đầu vào của mô hình dự đoán lỗi để tính độ đo F1-score và đường cong ROC. Dựa vào độ đo F1-score và ROC, có thể dùng đánh giá tính hiệu quả mỗi kỹ thuật lấy mẫu trong bài toán dự đoán lỗi phần mềm.

Để đạt được mục tiêu các kết quả đáng tin cậy và ổn định, trong các thực nghiệm, chiến lược xác thực chéo mười đoạn được sử dụng. Tuy nhiên, Fisher và cộng sự [13] cho rằng một số lượng lớn các thuật toán học bộc lộ các kết quả khác nhau nếu thứ tự lựa chọn các mẫu là khác nhau. Do vậy, xác thực chéo 10 đoạn được lặp lại 30 lần, mỗi lần thứ tự của các mẫu được xáo trộn ngẫu nhiên. Vì vậy, mỗi kết quả được trình bày trong phần này là trung bình của 30 lần chạy trên mỗi tập dữ liệu.

Bảy loại thuật toán phân lớp khác nhau được sử dụng trong thực nghiệm gồm Logistic Regression, K-nearest Neighbors, Naïve Bayes, Random Forest, Decision Tree, Support Vector Machine và Multilayer Perceptron.

Bảng 4. Tóm tắt 03 tập dữ liệu không cân bằng cao được sử dụng trong thực nghiệm

Tập dữ liệu	Thuộc tính	Mẫu	Mẫu lỗi	Mẫu không lỗi	% số mẫu lỗi
CLASS	21	11330	4227	7103	37,31 %
MC1	39	1988	46	1942	2,31 %
PC1	22	1109	77	1032	6,94 %

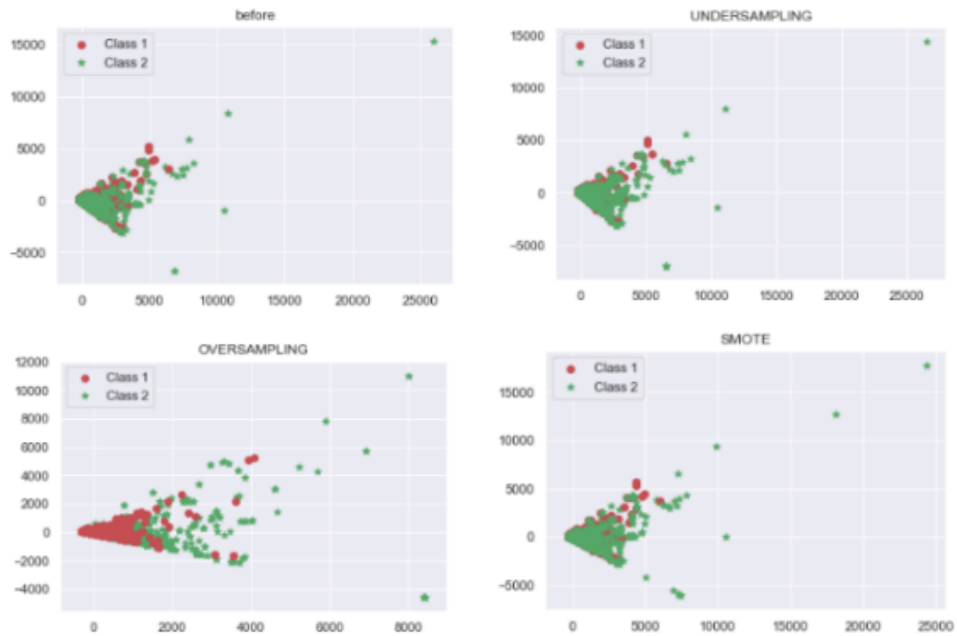
4.4. Kết quả thực nghiệm

Thực nghiệm này nhằm đánh giá việc sử dụng các kỹ thuật lấy mẫu dữ liệu ảnh hưởng đến độ chính xác của mô hình dự đoán lỗi. Với các tập dữ liệu khác nhau kết hợp với kỹ thuật lấy mẫu khác nhau dẫn đến các giá trị F1-score trung bình tốt hơn và đường cong ROC cận biên trên và biên trái hơn.

Hình 5 thể hiện các biểu đồ phân lớp của tập dữ liệu CLASS trước và sau khi sử dụng các kỹ thuật lấy mẫu. Các tập dữ liệu được phân lớp nhị phân:

o : Class 1: màu đỏ - thể hiện lớp không lỗi * : Class 2 : màu xanh – thể hiện lớp lỗi

Các biểu đồ được thể hiện lần lượt trước khi lấy mẫu (phía trên bên trái), được lấy mẫu bằng kỹ thuật Undersampling (phía trên bên phải), được lấy mẫu bằng kỹ thuật Oversampling (phía dưới bên trái), được lấy mẫu bằng kỹ thuật SMOTE (phía dưới bên phải).



Hình 5. Biểu đồ phân lớp tập dữ liệu CLASS trước và sau khi lấy mẫu

Dựa vào Bảng 4, tỷ lệ phần trăm mẫu lỗi chiếm trên tổng số mẫu ở tập dữ liệu CLASS là cao nhất. Điều này cho thấy mức độ cân bằng của tập dữ liệu này kém hơn so với các tập dữ liệu khác nên khi thực hiện lấy mẫu bằng kỹ thuật Undersampling ở tập này cho biểu đồ phân lớp gần giống với trước khi lấy mẫu nhất.

Đối với kỹ thuật Oversampling và SMOTE, kỹ thuật làm tăng tổng số lượng mẫu ở lớp thiểu số nên những mẫu có ký hiệu * màu xanh được thể hiện nhiều hơn so với ban đầu và Undersampling. Làm tương tự với 2 tập còn lại MC1 và PC1, Bảng 5 thể hiện các giá trị F1-score trung bình của 03 tập dữ liệu thực nghiệm không cân bằng trước và sau khi được lấy mẫu.

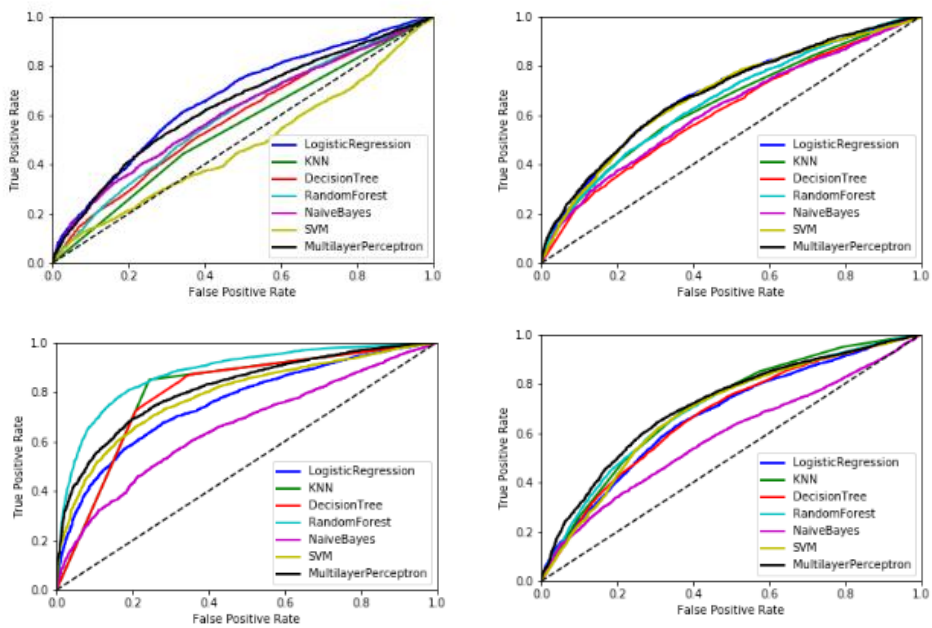
Bảng 5. Giá trị F1-score trung bình của các tập dữ liệu không cân bằng sử dụng các kỹ thuật lấy mẫu và không lấy mẫu

	Thuật toán	**Class**	**MC1**	**PC1**
Dữ liệu không cân bằng ban đầu	**LR**	0,3392	0,0733	0,1349
	KNN	0,4287	**0,3376**	**0,356**
	Tree	0,4195	0,1345	0,346
	Forest	0,4248	0,0594	0,2891
	NB	0,3161	0,1039	0,2709
	SVM	**0,4775**	0,2127	0,2114

	Thuật toán	Class	MC1	PC1
	MLP	0,4469	0,1466	0,2096
Under	**LR**	0,5883	0,6641	0,6402
	KNN	0,5777	**0,7507**	0,7693
	Tree	0,5477	0,5883	**0,778**
	Forest	0,5807	0,6713	0,766
	NB	0,3549	0,5286	0,5359
	SVM	**0,6206**	0,6867	0,6985
	MLP	0,6076	0,6785	0,701
Over	**LR**	0,6643	0,8081	0,7091
	KNN	**0,8145**	0,8192	**0,8752**
	Tree	0,7682	0,7735	0,8133
	Forest	0,8121	0,8021	0,8514
	NB	0,4193	0,6847	0,5859
	SVM	0,7136	0,7729	0,7658
	MLP	0,7343	**0,8238**	0,7792
SMOTE	**LR**	0,5959	0,7909	0,6456
	KNN	0,6675	0,9691	**0,8299**
	Tree	0,6316	0,9546	0,7191
	Forest	0,6604	**0,9811**	0,727
	NB	0,3497	0,7677	0,6013
	SVM	**0,6725**	0,9466	0,8064
	MLP	0,6584	0,9748	0,7777

Từ kết quả này, chúng ta có thể quan sát rằng khi kết hợp của các kỹ thuật lấy mẫu vào mô hình dự đoán thực hiện tốt hơn nhiều các mô hình không sử dụng lấy mẫu trên tất cả các tập dữ liệu thực nghiệm. Các giá trị trung bình F1-score khi kết hợp với các kỹ thuật lấy mẫu kết quả cao hơn so với dữ liệu mất cân bằng. Cũng từ Bảng 5, chúng ta dễ dàng nhận thấy kỹ thuật SMOTE cho kết quả tốt hơn Oversampling ở đa số các trường hợp thực nghiệm (trừ trường hợp khi sử dụng tập CLASS) và kỹ thuật Oversampling cho kết quả tốt hơn khi sử dụng kỹ thuật Undersampling.

Bước tiếp theo của thực nghiệm sau khi lấy mẫu là sử dụng mô hình dự đoán lỗi để tính độ đo F1-score và đường cong ROC. Hình 6 thể hiện các đường cong ROC.



Hình 6. Đường cong ROC sử dụng tập dữ liệu CLASS của các thuật toán trong việc dự đoán lỗi hướng phương thức. Mô hình dự đoán sử dụng dữ liệu theo thứ tự trước khi lấy mẫu dữ liệu, Undersampling, Oversampling và SMOTE

Quan sát Hình 6 kết hợp với Bảng 5, dễ dàng nhận thấy rằng, đối với mỗi tập dữ liệu, đường cong ROC đánh giá hiệu quả của các mô hình dự đoán gần tiệm cận với biên dọc trái và trên tuần tự khi sử dụng tập dữ liệu dữ liệu mất

cân bằng, dữ liệu được lấy mẫu với kỹ thuật Undersampling, Oversampling và SMOTE. Đường cong ROC gần tiệm cận với biên thì diện tích dưới đường cong AUC cũng theo thứ tự như trên. Điều này thể hiện tính hiệu quả của việc áp dụng các kỹ thuật lấy mẫu trong bài toán dự đoán lỗi phần mềm.

Thực nghiệm được thực hiện trên tập CLASS và cho kết quả độ đo F1-score của mô hình khi kết hợp với kỹ thuật Oversampling cho kết quả tốt hơn so với kỹ thuật Undersampling và SMOTE. Đường cong ROC thu được từ mô hình cũng cho kết quả cận biên trên và biên trái, diện tích phía dưới đường cong AUC cũng lớn hơn.

Dựa vào kết quả nghiên cứu được, chúng tôi đề xuất xếp hạng các kỹ thuật lấy mẫu dữ liệu cho mỗi tập dữ liệu để đánh giá mức độ hiệu quả của mỗi kỹ thuật khi kết hợp với mô hình dự đoán lỗi phần mềm.

Bảng 6. Bảng đánh giá mức độ hiệu quả của các kỹ thuật lấy mẫu

Tập dữ liệu	Undersampling	Oversampling	SMOTE
CLASS	1	3	2
MC1	1	2	3
PC1	1	3	2
Trung bình	1	2.67	2.33

Tóm lại, việc áp dụng các kỹ thuật lấy mẫu dữ liệu Undersampling, Oversampling và SMOTE đóng góp đến việc cải thiện đáng kể hiệu quả của các bộ phân lớp cơ sở và mô hình kết hợp cuối cùng. Các kết quả thu được đã chỉ ra vai trò quan trọng của tập dữ liệu cân bằng đối với hiệu quả dự đoán của từng thuật toán học máy.

V. KẾT LUẬN

Phân lớp dữ liệu mất cân bằng là một bài toán quan trọng và được ứng dụng vào nhiều lĩnh vực khác nhau trong thực tế, ảnh hưởng của vấn đề phân lớp mất cân bằng thường bị bỏ qua. Nhiều nghiên cứu [14] tập trung vào việc cải thiện độ chính xác của phân loại nhưng không xem xét vấn đề phân phối lớp mất cân bằng. Do đó, các bộ phân loại được xây dựng bởi các nghiên cứu này mất khả năng dự đoán chính xác cho các mẫu lớp thiểu số trong các bộ dữ liệu có số lượng mẫu lớp đa số lớn hơn nhiều so với số lượng mẫu lớp thiểu số. Những bài toán dự đoán lỗi phần mềm luôn liên quan đến vấn đề phân phối lớp mất cân bằng.

Những kỹ thuật lấy mẫu dữ liệu góp phần nâng cao hiệu suất để giải quyết bài toán này. Trong nghiên cứu này, chúng tôi đã trình bày những kỹ thuật lấy mẫu: Random Undersampling, Random Oversampling; và SMOTE; tiến hành các thực nghiệm để so sánh, đánh giá hiệu suất của các kỹ thuật trên 03 tập dữ liệu từ dự án của NASA. Nhìn chung các kỹ thuật tăng mẫu dữ liệu khi kết hợp với mô hình dự đoán lỗi đều cho kết quả tốt hơn kỹ thuật giảm mẫu.

Chiến lược của kỹ thuật Undersampling là làm giảm mẫu ở lớp đa số để cân bằng với lớp thiểu số, điều này dễ làm mất các mẫu quan trọng của lớp đa số nên nếu áp dụng kỹ thuật này vào với mô hình dự đoán sẽ cho độ chính xác dự đoán kém hơn. Ngược lại với Oversampling là làm tăng mẫu ở lớp thiểu số, số lượng và chất lượng mẫu được giữ nguyên nên cho kết quả tốt hơn Undersampling. Còn lại, chiến lược của SMOTE là tạo thêm một số lượng mẫu lỗi dựa trên các đặc tính của các mẫu được chọn, nó giúp mô hình sẽ có nhiều hơn cơ hội học từ các mẫu dữ liệu khác nhau. Nhưng, lợi ích SMOTE cũng chính là nhược điểm của kỹ thuật. SMOTE sẽ tạo ngẫu nhiên số lượng mẫu từ các thuộc tính của những mẫu có sẵn, nên nó sẽ tạo ra những trường hợp mẫu không có thực.

VI. LỜI CẢM ƠN

Nghiên cứu này được thực hiện từ nguồn kinh phí hỗ trợ của Bộ Giáo dục và Đào tạo trong đề tài có mã số B 2019-DNA-03.

TÀI LIỆU THAM KHẢO

[1] Le Hoang Son, Nakul Pritam, Manju Khari, Raghvendra Kumar, Pham Thi Minh Phuong, Pham Huy Thong, "Empirical Study of Software Defect Prediction: A Systematic Mapping", Symmetry 2019, 11, 212, 13 February 2019.

[2] Cagatay Rukshan Catal, Banu Diri, "A systematic review of software fault prediction studies", Expert Systems with Applications, Vol. 36, No. 4, pp. 7346-7354, May 2009.

[3] Akiyama, F., "An Example of Software System Debugging", Proceedings of the International Federation of Information Processing Societies Congress, 1971.

[4] Al-Qutaish, Rafa & Abran, Alain, "Halstead Metrics: Analysis of their Design", 2010.

[5] T. J. McCabe, "A Complexity Measure", IEEE Transactions on Software Engineering, Vols. SE-2, No. 4, pp. 308-320, Dec. 1976.

[6] Halstead, Maurice H, "Elements of Software Science", Amsterdam: Elsevier North-Holland, 1977.

[7] S.R. Chidamber; C. F. Kemerer, "A metrics suite for object oriented design", IEEE Transactions on Software Engineering, Vol. 20, No. 6, June 1994.

[8] T. Menzies, B. Turhan, A. Bener, G. Gay, B. Cukic, and Y. Jiang, "Implications of ceiling effects in defect predictors" in the 4th International Workshop on Predictor Models in Software Engineering, pp. 47-54, 2008.

[9] Nuno Moniz, Paula Branco, Luís Torgo, "Resampling Strategies for Imbalanced Time Series Forecasting," International Journal of Data Science and Analytics, Vol. 3, No. 3, pp. 161-181, May 2017.

[10] Chawla, Nitesh V; Herrera, Francisco; Garcia, Salvador; Fernandez, Alberto, "SMOTE for Learning from Imbalanced Data: Progress and Challenges, Marking the 15-year Anniversary", Journal of Artificial Intelligence Research, pp. 863-905, 2018.

[11] R. Akbani, S. Kwek, and N. Japkowicz, "Applying Support Vector Machines to Imbalanced Datasets", in Proceedings of 15th European Conference on Machine Learning, pp. 39-50, 2004.

[12] T. J. McCabe and C. W. Butler, "Design complexity measurement and testing", Communications of the ACM, 32(12), December 1989.

[13] D. H. Fisher, L. Xu, and N. Zard, "Ordering effects in clustering", in Proceedings of the 9th International Workshop of Machine Learning, pp. 162-168, 1992.

[14] Caragea, D., Cook, D., Honavar, V., "Gaining Insights into Support Vector Machine Pattern Classifiers Using Projection-Based Tour Methods", Proceedings of the KDD Conference, San Francisco, CA, pp. 251-256, 2001.

DEALING WITH UNBALANCED DATA IN SOFTWARE DEFECT PREDICTION PROBLEM

Le Song Toan, Nguyen Thanh Binh, Le Thi My Hanh, Nguyen Thanh Binh

ABSTRACT: *Predicting software defects helps to forecast potential faults of source code, reduces testing time, and increases product quality. Source code features are important information in prediction of software fault. However, many datasets for defect prediction are unbalanced, that means a big difference of data amount between classes. In this paper, we focus on data imbalanceproblems and the need for data sampling to process unbalanced data. We conduct the experiments with three data sampling techniques including: Random Undersampling, Random Oversampling and SMOTE. The techniques are applied to three NASA defect prediction datasets in the Promise respository. The results obtained show the efficiency of applying machine learning models in combination with sampling techniques to improve the accuracy of software defect prediction model.*