Cải Tiến Trọng số CFC và Rút Trích Đa Đặc Điểm Để Dò Tìm Những Báo Cáo Lỗi Trùng Nhau

Nhan Minh Phúc, Nguyễn Hoàng Duy Thiện
Khoa Kỹ thuật và Công nghệ, Đại học Trà Vinh
nhanminhphuc@tvu.edu.vn, thiennhd@2tvu.edu.vn

Dương Ngọc Vân Khanh
Khoa Kỹ thuật và Công nghệ, Đại học Trà Vinh
vankhanh@tvu.edu.vn

***Tóm tắt*—Trong những năm gần đây, nhiều dự án phần mềm sử dụng các hệ thống theo dõi lỗi để thu thập và xử lý các báo cáo lỗi. Khi các lỗi được báo cáo, một nhiệm vụ quan trọng là kiểm tra xem chúng có mới hay không, nếu chúng đã được báo cáo bởi một người dùng trước đó trong trường hợp này được gọi là những báo cáo lỗi trùng nhau. Một số kỹ thuật đã được đề xuất để tự động hóa phát hiện báo cáo lỗi trùng nhau và hầu hết trong số chúng dựa vào các kỹ thuật rút trích văn bản, hay sử dụng các báo cáo lỗi như các truy vấn. Để tiếp tục cải tiến hiệu suất từ phương pháp của họ, trong bài báo này, chúng tôi đề xuất một phương pháp mới để dự đoán các báo cáo lỗi trùng nhau. Cách tiếp cận của chúng tôi coi mô tả văn bản trong các báo cáo lỗi là các đặc điểm, khi đó sử dụng phương pháp rút trích 27 đặc điểm và cung cấp các đặc điểm này để cải thiện trọng số của CFC (Class Feature Centroid). Trong đánh giá sơ bộ, chúng tôi tiến hành thử nghiệm trên các bộ dữ liệu của nền tảng Eclipse, OpenOffice và Mozilla. Kết quả cho thấy phương pháp của chúng tôi được cải tiến so với các phương pháp được so sánh khoảng 6-12%.**

Từ khóa: Báo cáo lỗi; Trọng số; đa đặc điểm; CFC;

I. GIỚI THIỆU

Do sự phức tạp trong quá trình xây dựng nên hầu hết các phần mềm thường vẫn còn nhiều lỗi sau khi hoàn chỉnh. Những lỗi phần mềm đôi khi dẫn đến thiệt hại nhiều triệu USD [4]. Vì vậy việc xử lý lỗi trở thành một trong những vấn đề quan trọng cần thực hiện thường xuyên trong việc bảo trì phần mềm. Để giúp quản lý lỗi phần mềm và làm cho hệ thống đáng tin cậy hơn, những công cụ quản lý lỗi được xây dựng và ứng dụng vào các hệ thống lớn như Bugzilla, Eclipse,…những công cụ này cho phép người dùng sử dụng phần mềm như "tester" và gửi báo cáo lỗi mà họ phát hiện được đến hệ thống quản lý lỗi, thông tin này sau đó được tiếp nhận và xử lý để hoàn thiện độ tin cậy của phần mềm hơn.

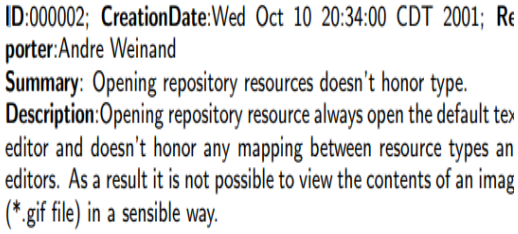
Mặt dù mang lại nhiều lợi ích trong việc cung cấp hệ thống báo cáo lỗi, nhưng nó cũng gây ra nhiều thách thức. Một trong những thách thức đó là cùng một lỗi được phát hiện bởi nhiều người dùng, khi đó có nhiều người gửi cùng một báo cáo lỗi đến hệ thống, gây nên tình trạng gọi là trùng lặp báo cáo lỗi. Điều này làm mất nhiều thời gian và công sức cho người phân loại, nghĩa là khi một báo cáo mới được gửi đến, họ phải kiểm tra xem báo cáo lỗi này đã được gửi đến trước đó chưa. Theo thống kê [2], [3] mỗi ngày có ít nhất 300 báo cáo lỗi được gửi đến hệ thống quản lý lỗi của Mozilla, số lượng này được xem là quá nhiều cho công việc phân loại. Vì vậy việc xây dựng một hệ thống tự động phân loại chẳng hay một báo cáo lỗi vừa được gửi đến đã được báo cáo trước đó hay chưa. Đây là chủ đề đang được các nhà nghiên cứu quan tâm hiện nay. Để giải quyết vấn đề những báo cáo lỗi trùng nhau, hiện tại trong cộng đồng nghiên cứu có hai hướng. Hướng thứ nhất khi có một báo cáo lỗi mới được gửi đến, sau đó xây dựng mô hình xử lý và kết quả trả về danh sách những báo cáo lỗi gần giống nhất với báo cáo lỗi được gửi đến trong top K. Phương pháp này được công bố bởi [1], [3], [4], [5]. Hướng thứ hai được công bố bởi Jalbert and Weimer [6] như sau, khi có một báo cáo mới được gửi đến, họ sẽ thực hiện việc phân loại thành hai nhóm, trùng nhau hay không trùng nhau, nghiên cứu này còn được gọi phân loại báo cáo lỗi bằng cách gán nhãn những báo cáo lỗi trùng nhau, và những báo cáo lỗi không trùng nhau. Theo thống kê bởi [9] hướng thứ nhất nhận được nhiều sự quan tâm hơn của các nhà nghiên cứu, ly do ngoài việc trả về kết quả top K danh sách báo cáo lỗi trùng nhau, nó gần như bao gồm luôn hướng thứ hai là phân loại báo cáo lỗi. Trong bài báo này chúng tôi cũng nghiên cứu theo phương pháp thứ nhất.

II. BÁO CÁO LỖI TRÙNG NHAU

Một báo cáo lỗi thông thường là một tập tin bao gồm vài thuộc tính như tóm tắt lỗi (summary), mô tả lỗi (description), dự án (project), người gửi (submitter), bình luận (comment)…Mỗi thuộc tính chứa những thông tin khác nhau. Ví dụ, thuộc tính summary dùng mô tả một cách ngắn gọn nội dung lỗi, trong khi thuộc tính description mô tả một cách chi tiết lý do phát sinh lỗi, các thao tác gây ra lỗi, cả hai thuộc tính này thường được mô tả theo dạng ngôn ngữ tự nhiên. những thuộc tính khác như project, comment…cũng phần nào hỗ trợ cho việc mô tả lỗi thêm rõ hơn.

Khoa kỹ thuật và Công nghệ, Đại học Trà Vinh
vankhanh@tvu.edu.vn

Trong công nghệ phần mềm, nhất là đối với các hệ thống phần mềm mã nguồn mở, hệ thống quản lý lỗi thường được mở cho người dùng thử nghiệm phần mềm, khi đó khó tránh khỏi trường hợp các người dùng khác nhau báo cáo cùng một lỗi giống nhau, này được gọi là những báo cáo lỗi trùng nhau. Hình 1 và hình 2 là một ví dụ về hai báo cáo lỗi trùng nhau trong kho phần mềm Eclipse. Trong đó hình 2 cho thấy mã báo cáo lỗi 009779 được thông báo trùng với báo cáo lỗi có mã số 000002. Thông thường khi một báo cáo lỗi mới vừa được gửi đến mà người phân loại xác


ID:000002; **CreationDate**:Wed Oct 10 20:34:00 CDT 2001; **Re**
porter:Andre Weinand
Summary: Opening repository resources doesn't honor type.
Description:Opening repository resource always open the default tex
editor and doesn't honor any mapping between resource types an
editors. As a result it is not possible to view the contents of an imag
(*.gif file) in a sensible way.

Hình 1: Một báo cáo lỗi trong dataset Eclipse

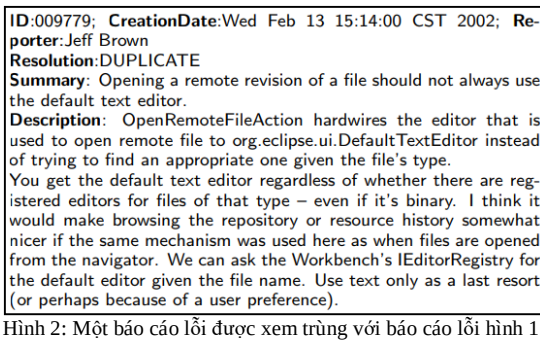
định báo cáo lỗi này bị trùng với những báo cáo đã được gửi trước đó thì báo cáo này sẽ được đánh dấu là trùng lặp (duplicate). Khi đó tất cả các báo cáo lỗi có cùng lỗi, chỉ duy nhất báo cáo lỗi đầu tiên trong số này không bị đánh dấu là trùng lặp. Trong bài báo này chúng tôi gọi báo cáo lỗi đó là báo cáo lỗi chính (*master*) và những báo cáo lỗi được gửi sau trùng với báo cáo lỗi này được gọi là trùng lắp của nó (*duplicates*).

III. KỸ THUẬT TRÍCH CHỌN ĐẶC ĐIỂM

Trong những kỹ thuật phân loại văn bản, hay xác định những văn bản tương tự nhau thì kỹ thuật chọn đặc điểm đóng vai trò quan trọng. Trong trường hợp xác định sự trùng lắp giữa hai báo cáo lỗi trong kho phần mềm mã nguồn mở cũng tương tự như vậy. Việc trích chọn đặc điểm tốt sẽ góp phần rất lớn vào việc xác định các báo cáo lỗi trùng nhau chính xác hơn. Trong phương pháp được giới thiệu, chúng tôi triển khai công thức bên dưới để đánh giá sự giống nhau (*sim*) giữa hai báo cáo lỗi.

$$sim\ (B_1, B_2) = \sum_{w\ \epsilon B_1 \cap B_2} idf(w) \qquad (1)$$

Trong (1) $sim\ (B_1, B_2)$ trả về kết quả giống nhau giữa hai túi từ B1 và B2. Sự giống nhau giữa hai báo cáo lỗi này được tính là tổng giá trị trọng số của những từ giống nhau trong IDF. Giá trị trọng số của mỗi từ trong báo cáo lỗi được tính dựa vào tất cả báo cáo lỗi trong kho dữ liệu. Lý do tại sao trong phương pháp của chúng tôi không sử TF*IDF cho việc tính trọng số từ mà sử dụng CFC. Theo [10] TF-IDF còn

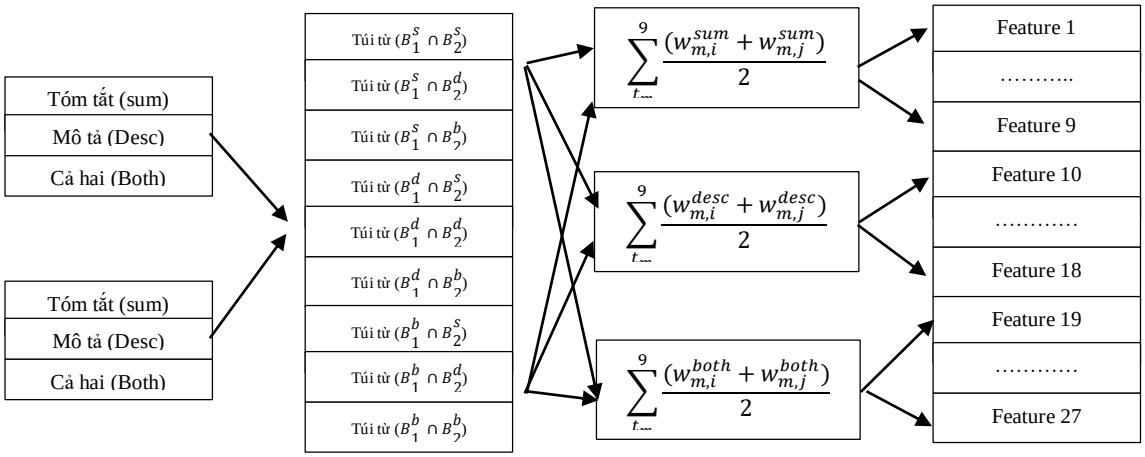

ID:009779; **CreationDate**:Wed Feb 13 15:14:00 CST 2002; **Re-porter**:Jeff Brown
Resolution:DUPLICATE
Summary: Opening a remote revision of a file should not always use the default text editor.
Description: OpenRemoteFileAction hardwires the editor that is used to open remote file to org.eclipse.ui.DefaultTextEditor instead of trying to find an appropriate one given the file's type.
You get the default text editor regardless of whether there are registered editors for files of that type – even if it's binary. I think it would make browsing the repository or resource history somewhat nicer if the same mechanism was used here as when files are opened from the navigator. We can ask the Workbench's IEditorRegistry for the default editor given the file name. Use text only as a last resort (or perhaps because of a user preference).

Hình 2: Một báo cáo lỗi được xem trùng với báo cáo lỗi hình 1

những hạn chế, trong khi [11] cho thấy những ưu điểm của nó trong việc hỗ trợ phân loại văn bản tốt, điều này giúp cho kết quả xác định độ tương đồng trong việc dò tìm báo cáo lỗi trùng nhau hiệu quả hơn. Ngoài ra chúng tôi cũng đã tiến hành thực nghiệm để kiểm chứng với phương pháp phổ biến đối với việc trích chọn đặc điểm là Fisher score [16] đã cho thấy, với 27 đặc điểm sử dụng *CFC*, chúng tôi nhận thấy kết quả tốt hơn trong việc xác định độ giống nhau giữa hai báo cáo lỗi so với khi kết hợp 27 đặc điểm với *TF*IDF*. Vì vậy, chúng tôi đã quyết định chọn CFC-27 cho việc tính trọng số đặc điểm trong phương pháp của chúng tôi. Nói cách khách, mỗi đặc điểm sau khi được trích chọn trong phương pháp của chúng tôi sẽ được tính sự giống nhau dựa vào những từ trong báo cáo lỗi R_1, với những từ trong báo cáo lỗi R_2. Kết quả là mỗi đặc điểm sau trích chọn thực sự là sự tương tự giữa hai túi từ của hai báo cáo lỗi R_1 và R_2 được thể hiện công thức bên dưới:

$f\ (R_1, R_2)$= *sim (những từ trong báo cáo R1, những từ trong báo cáo R2)* (2)

Quan sát từ những file báo cáo lỗi có thể thấy rằng một báo cáo lỗi bao gồm hai trường (field) quan trọng là trường tóm tắt (summary) và trường mô tả (description). Khi đó chúng tôi sử dụng ba túi từ từ một file báo cáo lỗi. Trong đó một túi từ sử dụng cho trường tóm tắt, túi thứ hai sử dụng cho trường mô tả, và túi thứ ba sử dụng cho cả hai trường (tóm tắt + mô tả). Ví dụ để rút trích một đặc điểm để so sánh từ hai hai báo cáo lỗi, chúng ta có thể tính độ tương tự giữa túi từ trong báo cáo lỗi thứ nhất của trường tóm tắt với túi từ trong báo cáo lỗi thứ hai trong trường mô tả. Tương tự như vậy chúng ta có thể tính sự giống nhau giữa những từ trong báo cáo lỗi từ cả hai trường tóm tắt và mô tả của một báo cáo lỗi này với trường tóm tắt của báo cáo lỗi khác, điều này cũng tương tự với sự kết hợp của những trường khác đối với cả hai báo cáo lỗi. Ngoài ra chúng tôi cũng tính để tách ra từ ba loại *IDF* trong kho báo cáo lỗi. Một là tập hợp từ tất cả trường tóm tắt, loại thứ hai từ tất cả trường

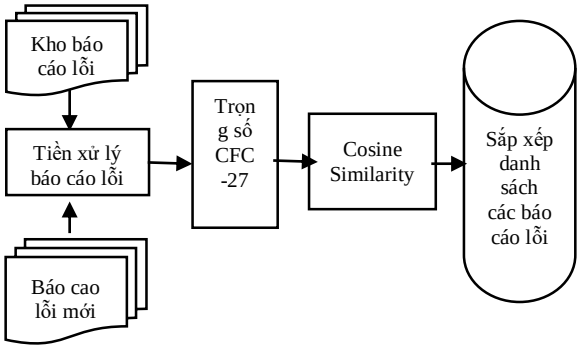


Hình 3: 27 đặc điểm dựa vào trọng số CFC

mô tả, và cuối cùng là từ cả hai trường tóm tắt và mô tả. Chúng tôi thể hiện ba loại IDF này theo quy ước tuần tự như IDF^{sum}, IDF^{desc}, và IDF^{both}. Kết quả của hàm f được định nghĩa trong (2) phụ thuộc vào lựa chọn túi từ đối với báo cáo lỗi R1, lựa chọn túi từ cho báo cáo lỗi R2 và lựa chọn tính *IDF*. Chúng tôi xem mỗi sự kết hợp như một đặc điểm khác nhau, vì vậy tổng số đặc điểm khác nhau được trích chọn là 3 x 3 x 3, nghĩa là có 27 đặc điểm có thể được trích chọn. Hình 3 cho thấy cách 27 đặc điểm được trích chọn từ hai báo cáo lỗi.

IV. NHỮNG NGHIÊN CỨU LIÊN QUAN

Một trong những người tiên phong đưa ra phương pháp giải quyết các vấn đề về báo cáo lỗi trùng nhau phải kể đến là Runeson et al. [5]. Trong phương pháp được giới thiệu, họ đã sử dụng phương pháp xử lý ngôn ngữ tự nhiên với kỹ thuật tách từ (tokenization), chuyển một từ về dạng gốc (stemming) và xóa bỏ những từ ít ý nghĩa (stop word removal). Những từ còn lại trong báo cáo lỗi được chuyển sang mô hình không gian vector (vector space), mỗi từ tương ứng một vector và được tính dựa vào trọng số của từ



Hình 4: Luồng xử lý dò tìm báo cáo lỗi trùng nhau

(weight (word)) theo công thức TF như sau:

$$Weight\ (word) = 1+log_2(TF(word)) \quad (3)$$

Phương pháp này cho kết quả đạt khoảng 40% với kho dữ liệu báo cáo lỗi Sony Ericsson Mobile Communications. Trong [6], Wang et al. cải tiến từ phương pháp của Runeson et al. sang hai hướng. Đầu tiên họ xem xét trọng số của từ không chỉ đối với TF mà còn cả IDF. Khi đó trọng số của từ được họ tính như sau:

$$Weight\ (word) = TF(word)*IDF(word) \quad (4)$$

Thứ hai họ xem xét thông tin thực thi của các báo cáo lỗi dẫn đến trùng nhau. Sau đó họ tính độ tương tự giữa hai báo cáo lỗi sử dụng cosine similarity. Kết quả thực nghiệm với dataset Firefox cho thấy kết quả đạt độ chính xác trong dò tìm các báo cáo lỗi trùng nhau từ 67-93%.

Alipour et al. [10] đã giới thiệu một kỹ thuật mới sử dụng thuật toán ra quyết định, phương pháp này đưa ra dự đoán dựa vào từng cặp báo cáo lỗi để xem họ

Procedure ProposeCandidates
Input:
Q: a new report
Rep: the bug repository
N: the expected size of candidate list
Output:
$result$: a list of N masters of which Q is a likely duplicate
Body:
1: $Candidates$ = an empty min-heap of maximum size N
2: **for each** bucket $B \in Buckets(Rep)$ **do**
3: $similarity = PredictBucket(Q, B)$
4: $Master(B).similarity = similarity$
5: add $Master(B)$ to $Candidates$
6: **end for**
7: sort $Candidates$ in descending order of field $similarity$
8: **return** sorted $Candidates$

Hình 5: Thuật toán tổng quát quá trình xử lý

có trùng nhau hay không? Trong kỹ thuật này họ sử dụng trọng số BM25F và thông tin văn bản trong file báo cáo lỗi phân loại theo lĩnh vực. Phương pháp này được đánh giá đạt tốt hơn 11.55% so với phương pháp Sun et al. [9] cho tập dữ liệu Android. Năm 2016, Meng-jie Lin at el. [11] đã giới thiệu chiến lược dò tìm dựa vào các đặc điểm tương quan. Trong đó xem xét các yếu tố liên quan dựa vào các đặc điểm khác nhau của báo cáo lỗi. Phương pháp này cho kết quả đạt gần 87 - 90% đối với tập dữ liệu Apache, ArgoUML và SVN.

V. PHƯƠNG PHÁP ĐA ĐẶC ĐIỂM

Phương pháp dò tìm tự động những báo cáo lỗi trùng nhau có thể được xem như một ứng dụng sử dụng kỹ thuật trích chọn thông tin và phân loại văn bản, mục đích của nó là cải thiện chất lượng phần mềm và giảm thời gian và chi phí cho người phát triển để phân loại cũng như xác định những báo cáo lỗi trùng nhau, nghĩa là những báo cáo lỗi đã được người dùng này gửi rồi, sau đó có người dùng khác gửi lại, trường hợp này gọi là trùng nhau. Trong phương pháp được giới thiệu, chúng tôi có sự thay đổi và cải tiến từ các phương pháp trước đây [5], [6]. Đầu tiên khi một báo cáo lỗi mới được gửi đến, hệ thống xử lý để phân loại báo cáo lỗi này sang hai lớp: trùng lắp và không trùng lắp, khi đó chúng tôi sẽ tính 27 loại đặc điểm khác nhau dựa vào độ tương tự giữa các báo cáo lỗi, và sử dụng những đặc điểm này cho việc tính trọng số đặc điểm để giúp xác định độ giống nhau giữa các báo cáo lỗi chính xác hơn. Hình 4 thể hiện luồng dữ liệu chính trong phương pháp dò tìm những báo cáo lỗi trùng nhau. Hình 5 cho thấy thuật toán xử lý tổng quát các bước thực hiện.

A. Tiền xử lý

Đây là bước đầu tiên cũng là bước quan trọng góp phần xác định độ chính xác của phương pháp dò tìm những báo cáo lỗi trùng nhau. Trong bước này chúng tôi sử dụng kỹ thuật trích chọn đặc điểm như đã giới thiệu phần 3, như hình 3. Đới với xử lý ngôn ngữ tự nhiên, chúng tôi theo Manning and Schütze [6], khi đó việc xử lý ngôn ngữ tự nhiên (NLP) trong một báo cáo lỗi được chia làm các bước sau:

- Tách từ (tokenization)
- Chuyển một từ về dạng gốc (Stemming)
- Xóa bỏ những từ ít ý nghĩa (stop words removal)

1) Tách từ

Tách từ là một kỹ thuật nhằm mục đích xác định ranh giới của các từ trong văn bản, cũng có thể hiểu đơn giản rằng tách từ chính là tách một đoạn text (một chuỗi liên tiếp các ký tự) thành những từ (*word* hay *token*) riêng lẻ và loại bỏ các dấu trong câu gây nhiễu, ví dụ dấu ngoặc, dấu nháy đơn, dấu nháy kép, dấu gạch nối…

2) Chuyển một từ về dạng gốc

Do báo cáo lỗi trong các kho phần mềm mã nguồn mở sử dụng ngôn ngữ tiếng anh, vì vậy mỗi từ trong báo cáo lỗi có thể được viết theo những dạng ngữ pháp khác nhau nhưng vẫn chứa cùng một thông tin. Do đó việc xử dụng stemming mục đích là để xử lý mỗi từ ở những dạng ngữ pháp khác nhau trở về từ gốc của nó, điều này giúp dễ dàng hơn trong việc tính toán và xác định những báo cáo lỗi trùng nhau. Ví dụ như từ *worded* và *working* sẽ được chuyển thành từ gốc là *work*. Những động từ cũng được chuyển trở lại nguyên mẫu của nó. Ví dụ *was* và *being* sẽ trở thành *be*.

3) Xóa bỏ những từ ít ý nghĩa

Thông thường trong một file báo cáo lỗi thường những từ hay thông tin dư thừa, hay nói chính xác hơn là chứa những từ mà bản thân nó không có nghĩa, hay những từ nối giống như *the, that, when, and, or*…những từ này không chứa thông tin cụ thể có ích cho việc xử lý tự động những báo cáo lỗi trùng nhau. Những từ này có nhiều trong các file báo cáo lỗi và nó hầu như không liên quan đến nội dung cụ thể nếu không loại bỏ nó sẽ ảnh hưởng rất nhiều đến việc xác định sự giống nhau giữa các file báo cáo lỗi. Thông thường việc xử lý những từ này bằng cách liệt kê một danh sách những từ không có nghĩa hay còn gọi là không có ích cho việc xử lý. Danh sách này thường gọi là danh sách "stop words", khi đó những báo cáo lỗi có chứa những từ trong danh sách này sẽ bị loại bỏ.

B. Tính trọng số của từ CFC-27

Phương pháp phổ biến nhất trong việc tính trọng số của từ là chuyển đổi văn bản sang mô hình không gian vector và tính TF-IDF. Nó là một phương pháp thống kê để đánh giá tầm quan trọng của một từ trong văn bản và được định nghĩa như sau:

$$idf(term) = log_2(\frac{D_{all}}{D_{term}}) \qquad (5)$$

Trong (1), D_{all} là số báo cáo lỗi trong kho báo báo lỗi, D_{term} là số báo cáo lỗi có chứa từ đó trong báo cáo lỗi. Nghĩa là đối với một từ trong báo cáo lỗi, nếu nó càng ít xuất hiện trong các báo cáo lỗi, thì nó càng có ý nghĩa quan trọng trong việc phân loại các báo cáo lỗi.

Tuy nhiên theo [12] TF-IDF còn nhiều hạn chế, và dựa vào [12] đã cho thấy được ưu điểm của CFC trong việc phân loại văn bản. Chúng tôi đã quyết định điều chỉnh cách tính trọng số trong phương pháp này với

thực nghiệm TF-IDF, IDF-IDF-27 và CFC-27. Trong CFC, trọng lượng của từ w_{ij} được tính như sau:

$$w_{ij} = b^{\frac{DF_{t_i}^j}{|C_j|}} \times \log(\frac{|C|}{CF_{t_i}}) \qquad (6)$$

Trong đó t_i là từ (term) trong báo cáo lỗi, $DF_{t_i}^j$ là số báo cáo lỗi chứa t_i của lớp C_j,$|C_j|$ là số báo cáo lỗi trong lớp C_j, $|C|$ là tổng số lớp, CF_{t_i} là số lớp chứa t_i, và b là tham số lớn hơn một, dùng để điều chỉnh cho trọng lượng w_{ij}. Trong CFC, $b^{\frac{DF_{t_i}^j}{|C_j|}}$ xem xét đến số báo cáo lỗi chứa mức độ xuất hiện thường xuyên của một từ bên trong lớp. Công thức log xem xét mức độ giống như IDF truyền thống.

C. *Tính độ tương đồng giữa hai báo cáo lỗi*

Trong bước này sẽ tiến hành xác định sự tương đồng giữa các báo cáo lỗi, khi có một báo cáo lỗi mới được gửi đến. Độ tương đồng của hai báo cáo lỗi BR_i và BR_j được tính dựa vào việc trích chọn từ 27 đặc trưng cùng với sự cải tiến trong cách tính trọng số CFC-27 như sau:

$$Cos\left(\overrightarrow{BR_i}, \overrightarrow{BR_j}\right) = \frac{\overrightarrow{BR_i} . \overrightarrow{BR_j}}{|\overrightarrow{BR_i}| . |\overrightarrow{BR_j}|}$$

VI. KẾT QUẢ THỰC NGHIỆM

A. *Môi trường thực nghiệm*

Chúng tôi đã tiến hành thực nghiệm với ba kho báo cáo lỗi của những dự án phần mềm mở là Mozilla, OpenOffice, và Eclipse. Thống kê chi tiết về 3 kho phần mềm này được mô tả trong bảng 6.1.

Bảng 6.1 Thông tin về datasets

Kho báo cáo lỗi	Thời gian	Số lượng báo cáo lỗi	Số lượng trùng
Mozilla	01/2010-12/2010	75.653	6.925
OpenOffice	01/2008-12/2010	31.138	3.171
Eclipse	01/2008-12/2008	45.234	3.080

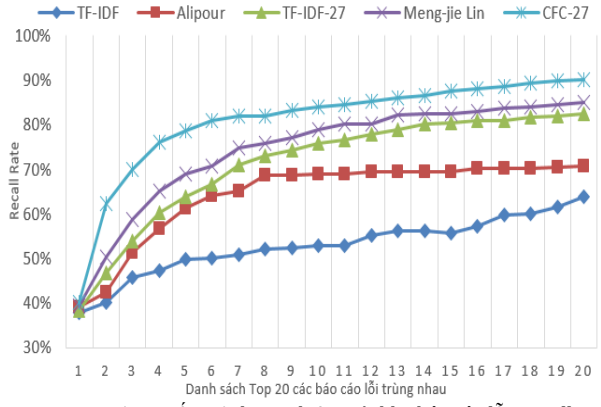
A. *Phương pháp đánh giá*

Để đánh giá phương pháp dò tìm, chúng tôi sử dụng đơn vị đo lường gọi là Recall rate, phương pháp này được những công bố trước sử dụng cho phương pháp dò tìm báo cáo lỗi trùng nhau, nó được tính dựa trên bao nhiêu báo cáo lỗi có thể được dò tìm đúng trong danh sách những báo cáo lỗi trùng nhau và nó được định nghĩa như sau:

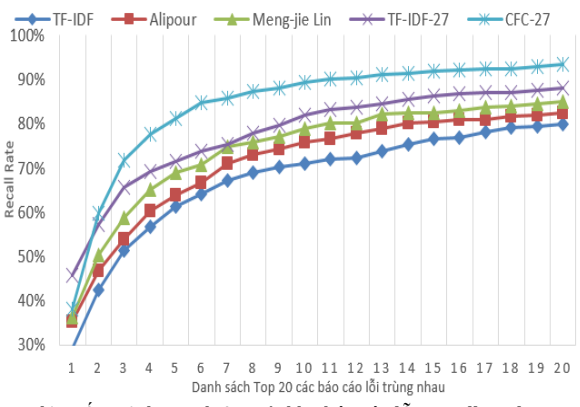
$$\text{Recall rate} = \frac{\text{Số những dự đoán đúng}}{\text{Tổng số những báo cáo lỗi trùng nhau}}$$

B. *Nghiên cứu kết quả thực nghiệm*

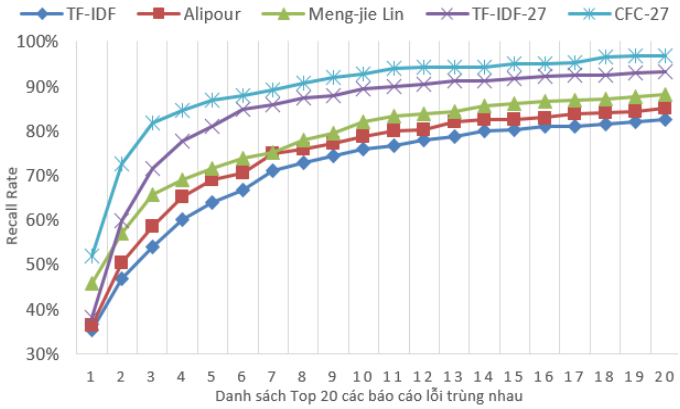
Để thấy sự hiệu quả của phương pháp được giới thiệu dựa vào trọng số CFC kết hợp với rút trích 27



a) Kết quả thực nghiệm với kho báo cáo lỗi Mozilla



b) Kết quả thực nghiệm với kho báo cáo lỗi Mozilla Eclipse



c) Kết quả thực nghiệm với kho báo cáo lỗi Mozilla OpenOffice

Hình 6: Kết quả so sánh CFC-27 với các phương pháp khác

đặc điểm (CFC-27), chúng tôi tiến hành so sánh nó với phương pháp truyền thống tính trọng số dựa vào TF-IDF và với sự kết hợp của TF-IDF với rút trích 27 đặc điểm (TF-IDF-27). Chúng tôi cũng so sánh với phương pháp của Alipour, và Meng-jie Lin. Kết quả thực nghiệm cho thấy phương pháp được giới thiệu cải tiến rõ rệt trong tất cả ba dự án. Hình 6 chỉ ra sự cải thiện đáng kể của CFC-27 so với các phương pháp được so sánh.

VII. KẾT LUẬN

Việc dò tìm trùng nhau của những báo cáo lỗi là một trong những vấn đề quan trọng trong việc bảo trì phần mềm trong những năm gần đây. Trong bài báo này chúng tôi giới thiệu một phương pháp dò tìm dựa vào trọng số mở rộng và rút trích đa đặc điểm (CFC-27) để cải tiến việc thực thi dò tìm những báo cáo lỗi trùng nhau. Kết quả thực nghiệm từ ba dự án mã nguồn mở cho thấy phương pháp này mang lại hiệu quả cao trong việc dò tìm các báo cáo lỗi trùng nhau, đặc biệt là khi so sánh với các phương pháp được giới thiệu trước đây, phương pháp CFC-27 đã cho kết quả tốt hơn và hiệu quả hơn trong việc dò tìm những báo cáo lỗi trùng nhau khoảng 6-12% so với các phương pháp trước đó.

TÀI LIỆU THAM KHẢO

[1] J.Sutherland, "Business objects in corporate information systems," in *In ACM Computing Surveys*, 2006.

[2] L. Hiew, "Assisted Detection of Duplicate Bug Reports," in *Master Thesis, The University of British Columbia, May 2006*, The University of British Columbia, 2006.

[3] L. H. a. G. C. M. John Anvik, "Coping with an Open Bug Repository," in *Proceedings of the OOPSLA workshop on Eclipse technology eXchange*, LA, USA, 2005.

[4] M. A. a. O. N. Runeson, "Detection of Duplicate Defect Reports using Natural Language Processing," in *in Proceedings of the 29th International Conference on Software Engineering (ICSE 2007),ACM*, pp. 499–510., 2007.

[5] L. Z. T. X. J. A. J. S. Xiaoyin Wang, "An approach to detecting duplicate bug reports using natural language and execution information,IEEE, ACM," in *In Proceedings of the 30th international conference on Software engineering*, pp. 461-470, 2008.

[6] C. Sun, D. Lo, X. Wang, J. Jiang and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in *in Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, ACM*, pp. 45-54. , 2010.

[7] D.-D. H. Z.-Y. F. C.-Z. Y. Chao-Yuan Lee, "Mining Temporal Information to Improve Duplication Detection on Bug Reports," in *Advanced Applied Informatics (IIAI-AAI) 2015 IIAI 4th International Congress on 2015*, pp. 551-555, Taiwan, 2015.

[8] W. W. Nicholas Jalbert, "Automated Duplicate Detection for Bug Tracking Systems," in *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)* , Anchorage, AK, USA, 2008.

[9] Y. C. Tian, "Improved duplicate bug report identification," in *In proceeding of the 16th European Conference on Software Maintenance and Reengineering*.

[10] W. T. Xia Tian, "An improvement to TF: Term Distribution based," in *2010 Second International Conference on Networks Security, Wireless Communications and Trusted Computing*, 2010.

[11] J. y. Z. a. M. y. G. Hu Guan, "A Class-Feature-Centroid Classifier for Text Categorization," in *in Proceedings of the18th International Conference on World Wide Web,IEEE*, Marid, 2009.

[12] A. H. a. E. S. Alipour, "A contextual approach towards more accurate duplicate bug report detection," in *10th Working Conference on Mining Software Repositories (MSR)*, San Francisco, CA,, pp. 183-192., 2013.

[13] L. Z. Y. C. Y. C. Meng-Jie, "Enhancements for duplication detection in bug reports with manifold correlation features," *Journal of Systems and Software, Elservier*, vol. Volume 121, no. November, pp. Pages 223-233, 2016.