ارائه یک روش جدید مبتنی بر مدل برداری جهت پیشگویی خطاهای احتمالی یک نرم‌افزار بر اساس یادگیری از منابع دیگر

حسن اسمخان[1]، مجید دلیری[2]

[1] دانشجوی درس تکامل نرم‌افزار دانشکده کامپیوتر، شماره دانشجویی ۹۷۳۰۱۷۸۶
h.ismkhan@ce.sharif.edu یا esmkhan@gmail.com

[2] دانشجوی درس تکامل نرم‌افزار دانشکده کامپیوتر، شماره دانشجویی ۹۷۲۱۳۰۰۳
majiddaliri35@gmail.com

چکیده

همان‌طور که می‌دانیم پیشگویی و کشف خطا تأثیر چشمگیری در کاهش هزینه تولید نرم‌افزار دارد. باوجودآنکه الگوریتم‌های زیادی جهت پیشگویی و کشف خطا ارائه‌شده است، بااین‌حال اگر نتوان گفت همه آن‌ها، اکثر آن‌ها بر پایه نسخه‌های قبلی کدهای مربوط به نرم‌افزار بنیان نهاده شده‌اند. به این صورت که با یادگیری خطا از نسخه‌های قبلی، احتمال و محل وجود خطا در نسخه‌های بعدی را پیش‌بینی می‌کنند. به همین جهت بررسی خطا در نسخه اولیه نرم‌افزار غیرممکن یا حداقل پیچیده‌تر می‌شود. این مقاله پیاده‌سازی مدل جدید برداری ارائه‌شده توسط این گروه در فاز قبل را توضیح داده و به بررسی نتایج آزمایش‌های انجام‌شده در خصوص ارزیابی کارایی آن می‌پردازد. در بررسی کارایی الگوریتم پیشنهادی پیاده‌سازی شده، از یک مجموعه مشتمل بر حدود ۳۰،۰۰۰ فایل برنامه جاوا در قالب ۶ پروژه مختلف استفاده می‌شود.

کلمات کلیدی

پیش‌بینی خطا؛ پیش‌بینی خطا در پیاده‌سازی؛ خطای نسخه اولیه.

۱- مقدمه

الگوریتم‌های فراوانی جهت پیشگویی خطا ارائه‌شده‌اند [1, 2]. اما جهت اعمال اکثر آن‌ها نیاز است تا چند نسخه پیشین از نرم‌افزار در دسترس بوده و الگوریتم خطایاب موردنظر با در نظر گرفتن آن‌ها و یادگیری ایرادات تصحیح‌شده، بتواند تا حدی خطاهای نسخه فعلی را پیش‌بینی کند. برخی از این روش‌ها سعی در افزایش دقت و سرعت این فرآیند دارند [3, 4]، برخی موارد خطای انسانی را موردمطالعه قرار می‌دهند [2] و برخی نیز در خصوص محل تصحیح خطا توصیه‌های خاصی ارائه می‌دهند [5]. اما نکته قابل‌توجه این است که اکثر این روش‌ها یا حداقل موارد اشاره‌شده در مراجع این مقاله هیچ‌کدام در خصوص پیشگویی خطا به هنگام پیاده‌سازی نسخه اولیه نرم‌افزار راهکاری ارائه نمی‌دهند حال‌آنکه می‌دانیم همین خطاها در نسخه اولیه نیز هزینه‌های زیادی را تحمیل می‌کند.

اعضای این گروه در این پروژه قصد دارند بررسی کنند که آیا می‌توان با بررسی خطاهای موجود در دیگر پروژه‌ها، از تجارب موارد مشابه در پیاده‌سازی اولیه یک نرم‌افزار دیگر نیز استفاده کرد؟ طوری که از همان ابتدا و در همان نسخه اولیه با ارائه برخی توصیه‌ها به برنامه‌نویس از تکرار خطاهای مشابه جلوگیری کرده و هزینه‌ها را از همان ابتدا کاهش داد. جهت نیل به این هدف، این گروه روش پیشنهادی ارائه‌شده در فاز قبلی این پروژه را پیاده‌سازی کرده است که ادامه این مقاله به شرح نحوه پیاده‌سازی و بیان نتایج آزمایش‌های انجام‌شده در خصوص اعتبار سنجی روش پیشنهادی می‌پردازد.

۲- شرح پیاده‌سازی راه‌حل پیشنهادی

راهکار پیشنهادی در قالب یک ابزار کاربردی jar فایل ارائه می‌شود. این ابزار هم فاز یادگیری و هم فاز پیشگوئی را شامل می‌شود. ادامه این بخش در دو قسمت ارائه می‌شود. اولین قسمت به توضیح کدهای پیاده‌سازی و چگونگی باز استفاده از آن می‌پردازد. دومین قسمت به توضیح چگونگی استفاده از ابزار jar فایل حاصل‌شده می‌پردازد.

۲,۱- توضیح کدهای پیاده‌سازی و نحوه باز استفاده از آن

پیاده‌سازی مرحله یادگیری گام‌های متعددی را شامل می‌شود که از آن جمله می‌توان به موارد زیر اشاره کرد:

۱. خواندن فایل جاوا.
۲. استخراج کلمات.
۳. اعمال الگوریتم‌های پردازش زبانی و استخراج فرم ساده‌شده هر کلمه.
۴. استخراج گراف با در نظرگیری کلمات مجاور [6, 7, 8] .
۵. اعمال الگوریتم PageRank [9] و حصول وزن هر کلمه.
۶. تشکیل یک بردار از طریق وزن‌های حاصل‌شده مرحله قبل.
۷. با توجه به این‌که فایل سالم یا خطادار باشد، بردار حاصل، به مرکز ثقل یکی از مجموعه‌های خطادار یا سالم اضافه‌شده و مرکز ثقل مربوطه بروز رسانی می‌شود.

مرحله پیشگوئی یک فایل نیز در ۶ گام اول دقیقاً مشابه مرحله یادگیری است، اما در گام ۷ صرفاً به بررسی فاصله بردار حاصل‌شده با مراکز ثقل دو مجموعه سالم و خطادار می‌پردازد. اگر بردار حاصل‌شده به مرکز ثقل فایل‌های خطادار نزدیک باشد، خطادار تشخیص داده می‌شود.

تمامی گام‌های هر دو مرحله یادگیری و پیشگوئی در قالب ۵ کلاس جاوا شامل code2Tokens_، graph2Weights_، _tokens2Graph ، KeyWords_Java و MyPredictor پیاده‌سازی می‌شود، با این توضیح که هر کلاس در داخل فایل از نوع java. بانام مشابه قرار دارد.

گام‌های ۱ الی ۳ در کلاس code2Tokens_ پیاده‌سازی می‌شود. این کلاس شامل تابع ایستای apply است که بعد از اعمال گام‌های ۱ الی ۳ از طریق ابزار samurai [10]، نتیجه را در قالب یک لیست آرایه‌ای[1] از نوع رشته[2] برمی‌گرداند.

```
while( (line=br.readLine()) != null) {
    srcLines.add(line);

    SamuraiSplitter ssplitter =
            new SamuraiSplitter(srcLines);
    HashMap<String, String> hmap =
            ssplitter.getSplittedTokenMap();

    for (String key : (Iterable<String>)hmap.keySet()) {
        fromSamurai.add((String)hmap.get(key));
    }
    srcLines.clear();
}
```

شکل (۱) : کد استخراج کلمات از کد با ابزار samurai

شکل (۱) نحوه استفاده از این ابزار را در استخراج و پیش‌پردازش کلمات نشان می‌دهد. در هر اجرای حلقه while یک خط از منبع کد خوانده‌شده و به ابزار SamuraiSplitter تحویل داده می‌شود. این ابزار متن پردازش‌شده را در قالب یک HashMap برمی‌گرداند. البته قابل‌ذکر است، بعد از اعمال تابع apply کلاس code2Tokens_ ، حروف اضافه و کلمات کلیدی از طریق تابع isStopWord و isKeyWord کلاس KeyWords_Java صافی شده و به گام بعدی انتقال داده نمی‌شوند.

برای گام ۴، یعنی تشکیل گراف که در کلاس tokens2Graph_ انجام می‌شود از ابزار jgraph استفاده می‌شود. در پیاده‌سازی این پروژه از کلمه مجاور بعدی و کلمه مجاور قبلی در تشکیل گراف استفاده می‌شود. شکل (۲) قسمت اصلی این پیاده‌سازی را نشان می‌دهد.

```
DirectedGraph<String, DefaultEdge> graph =
        new DefaultDirectedGraph<>(DefaultEdge.class);
for (String term : terms)
    graph.addVertex(term);
for (int i = 1; i < terms.size() - 1; i++) {
    graph.addEdge(terms.get(i), terms.get(i + 1));
    graph.addEdge(terms.get(i), terms.get(i - 1));
}
if (terms.size() > 1) {
    graph.addEdge(terms.get(0), terms.get(1));
    graph.addEdge(terms.get(terms.size() - 1),
            terms.get(terms.size() - 2));
}
return graph;
```

شکل (۲) : تشکیل گراف بر اساس کلمات مجاور

گام ۵ و ۶ با بهره‌گیری از ابزار ca.usask.cs.srlab.pagerank پیاده‌سازی می‌شود. این ابزار به متعلق به آزمایشگاه تحقیق در نرم‌افزار[3] دانشگاه Saskatchewan تعلق دارد. سمت اصلی کد در شکل (۳) نمایش داده می‌شود.

```
HashMap<String, Double> tokendb =
        new HashMap<>(graph.vertexSet().size());

PageRankProvider ranker =
        new PageRankProvider(graph, tokendb);
ranker.calculatePageRank();
```

شکل (۳) : محاسبه وزن گره‌های گراف با الگوریتم PageRank

نهایتاً الگوریتم پیشنهادی در کلاس MyPredictor پیاده‌سازی می‌شود. API عمومی این کلاس به فرم زیر است:

۱. addToLearn
   - انواع ورودی: رشته و بولی[4]
   - نوع برگشتی: ندارد

[1] ArrayList
[2] String
[3] Software Research Lab
[4] Boolean

- توضیح: این تابع مسیر فایل ارائه‌شده برای یادگیری را در قالب یک رشته، و خطادار بودن آن را در قالب متغیر بولی گرفته و به مجموعه یادگیری خود اضافه می‌کند.

۲. predictIsBuggy
- نوع ورودی: رشته
- نوع برگشتی: بولی
- توضیح: این تابع یک رشته را به‌عنوان مسیر فایل گرفته و اگر خطادار تشخیص داده شود، مقدار True و در غیر این صورت مقدار false را برمی‌گرداند.

توابع فوق فقط دو تابع مهم کلاس MyPredictor را بیان می‌کند. علاوه بر موارد فوق، این کلاس شامل توابعی برای تهیه فایل پشتیبان از فایل‌های یادگیری شده، تابعی برای بارگذاری فایل‌های پشتیبان موجود، تابعی برای رتبه‌بندی فایل‌های ارائه‌شده بر مبنای میزان خطادار دار بودن و مواردی از این قبیل می‌شود.

جدول (۱) ابزارهای استفاده‌شده در پیاده‌سازی

خلاصه کارکرد	مرجع	نام ابزار
پیش‌پردازش-های زبانی	[10]	samurai
ساخت گراف	در قسمت Meta-Inf عنوان‌شده است که به Sun Microsystems Inc. تعلق دارد.	jgraph
پیاده سازی الگوریتم PageRank	آزمایشگاه SRLAB دانشگاه Saskatchewan	ca.usask.cs.srlab.pagerank

۲,۲- توضیح نحوه بهره‌گیری از ابزار jar فایل فراهم‌شده

این ابزار از طریق خطوط فرمان (CL) با فراهم کردن مسیر دو پوشه قابل بهره‌برداری است. با همین فرمان ساده امکانات زیر برای کاربر فراهم است:

۱. در نظر گرفتن پوشه حاوی فایل‌های کد جاوا برای یادگیری.
۲. امکان ایجاد فایل پشتیبان از فایل‌های یادگیری شده.
- ***توجه*** به این نکته ضروری است که فایل پشتیبان در این ابزار الزاماً باید با پسوند ser. تعریف شود.

۳. امکان بارگذاری فایل‌های پشتیبان قبلی و استفاده مجدد آن‌ها در پیشگویی.
۴. امکان پیشگوئی پوشه حاوی فایل‌های جاوا.

با این مقدمه، استفاده از این ابزار در محیط cmd سیستم‌عامل Windows صرفاً با یک دستور به فرم زیر صورت می‌گیرد:

```
java –jar predictor.jar folder path
```

در این دستور folder مسیر پوشه‌ای را نشان می‌دهد که حاوی فایل‌های جاوا یا فایل‌های پشتیبان (ser.) یا هر دو نوع است. اگر پوشه folder شامل حتی یک فایل جاوا باشد، در آن صورت الزاماً آن فولدر حتماً باید شامل فایل ***buggylist.tx***t_ نیز باشد. توجه داشته باشید که buggylist.txt_ شامل لیست فایل‌های جاوای خطادار موجود در folder خواهد بود. برای path، در ابزار، دو حالت در نظر گرفته‌شده است. path یا مسیر یک پوشه است که الزاماً باید موجود باشد یا مسیر یک فایل با پسوند ser. است که الزاماً هم وجود ندارد. برای path اگر مسیر پوشه نبوده و مسیر یک فایل با پسوند ser. باشد به این معناست که ابزار از آموخته‌های موجود در folder یک فایل پشتیبان با مسیر و نام ذکرشده در path می‌سازد. اما اگر path مسیر یک پوشه باشد، به این معناست این پوشه حاوی فایل‌هایی برای پیشگویی کردن است.

۳- ارزیابی راه‌حل پیشنهادی

این بخش با گزارش نتایج آزمایش‌های انجام‌شده به سؤالات پژوهشی زیر پاسخ خواهد داد:

سؤال پژوهشی ۱: آیا ابزار ارائه‌شده با ابزارهای مشابه موجود قابل‌رقابت است؟

سؤال پژوهشی ۲: آیا ابزار ارائه‌شده، سرعت قابل قبولی در پیشگوئی دارد؟

سؤال پژوهشی ۳: آیا ابزار ارائه‌شده امکان اعمال روی پروژه مشابه را دارد؟ ازآنجایی‌که این ابزار با این هدف پیاده‌سازی شده است تا بتواند با یادگیری از دیگر پروژه‌ها روی پروژه فعلی اعمال شود، این سؤال ابتدایی می‌تواند مطرح باشد که آیا این ابزار از طریق تاریخچه فایل‌های یک پروژه می‌تواند خطادار بودن فایل‌های فعلی درحال‌توسعه را تشخیص دهد.

در ادامه قبل از ارائه نتایج و پاسخ به سؤالات پژوهشی مجموعه داده‌ای مورداستفاده، معیارهای ارزیابی و دلایل انتخاب آن‌ها بحث شده و سپس در زیر بخش‌های بعدی این بخش بابیان نتایج آزمایش‌ها به سؤالات پژوهشی مطرح‌شده پاسخ‌خواهیم داد.

۳-۱- مجموعه داده‌ای

این مجموعه داده شامل حدود ۳۰۰۰۰ فایل جاوا در قالب شش پروژه مختلف بانام‌های ecf، eclipse.jdt.core، eclipse.jdt.debug، eclipse.jdt.ui، eclipse.pde.ui و tomcat70 است. در بدو جمع‌آوری این مجموعه داده‌ای از زیر دامنه‌های github کارهای تحقیقی معتبر، لیست فایل‌ها و فایل‌های خطادار در فرم مناسبی نبودند. در این کار تحقیقی فرم خاصی به آن‌ها داده شد. همه فایل‌های هرکدام از پروژه‌ها در یک پوشه بانام همان پروژه جمع شده و فایل‌های

خطادار موجود در آن پروژه در فایلی بنام buggylist.txt_ لیست شدند. جدول (۲) اطلاعاتی را در خصوص تعداد فایل و تعداد فایل‌های خطادار و درصد خطادار بودن یک پروژه را نشان می‌دهد.

جدول (۲) اطلاعات پروژه‌ها

	نام پروژه	تعداد کل فایل	تعداد فایل های خطادار	درصد فایل های خطادار
۱	ecf	۲۸۰۲	۱۴۱۴	٪۵۰
۲	eclipse.jdt.core	۵۹۰۸	۱۲۰۶	۲۰٪
۳	eclipse.jdt.debug	۱۵۳۱	۵۵۶	۳۶٪
۴	eclipse.jdt.ui	۱۰۹۲۷	۳۴۴۳	۳۲٪
۵	eclipse.pde.ui	۵۳۳۴	۲۰۶۶	۳۹٪
۶	tomcat70	۱۸۴۱	۱۱۱۱	۶۰٪

۳-۲- معیارهای ارزیابی

در بحث معیارهای ارزیابی توجه به این نکته حائز اهمیت است که باید بین پیشگویی خطا و خطایابی تفاوت قائل شد. در خطایابی، بعد از انتشار نرم‌افزار، وقتی خطایی رخ می‌دهد، از طریق گزارش آن خطا، هدف آن است که الگوریتم یابنده، فایل‌های خطادار را از بین انبوهی از فایل‌ها شناسایی کند. در چنین شرایطی معیارهایی مانند Top@N اینکه چه درصدی از فایل‌های خطادار را می‌توان با بررسی چند جواب اول الگوریتم یابنده می‌توان پوشش داد، یا MRR که در آن توانایی الگوریتم یابنده در شناسایی اولین (مهم‌ترین و مرتبط‌ترین) عامل خطا سنجیده می‌شود [11, 12, 13, 14]. موارد مشابه دیگری از این قبیل می‌توان ذکر کرد اما توجه داشته باشید که چنین معیارهایی برای ارزیابی الگوریتم‌های خطایاب مورد بهره قرارگرفته است نه برای الگوریتم‌های پیش‌بینی کننده.

در الگوریتم‌های پیش‌بینی کننده هدف آن است تا پیش‌بینی شود که یک فایل ارائه‌شده خطادار است یا نه. حال الگوریتم پیشگو یک فایل خطادار واقعی را خطادار پیشگویی کند True Positive (TP) یا یک فایل بدون خطا را به‌درستی بدون خطا تشخیص دهد True Negative (TN) معیار مثبتی برای آن پیشگو بوده، اما اگر به‌اشتباه فایل خطادار را سالم False Negative (FN) و یا فایل سالم را خطادار False Positive (FP) پیشگویی کند امتیاز منفی برای آن پیشگو به شمار می‌آید. ازجمله معیارهایی که این معیارها را می‌توانند برای پیشگو در مواجهه با چندین فایل پوشش دهند، می‌توان به precision و recall دو معیار سنتی در بازیابی اطلاعات اشاره کرد که هنوز نیز در کارهای تحقیقی معتبر از مقاله کنفرانسی [15] تا مقاله [2] از مجله معتبر IEEE TSE از آن ها بهره‌گیری می‌شود.

۳-۳- جواب سؤال پژوهشی ۱: آیا ابزار ارائه‌شده در این مقاله توان رقابت با ابزارهای مشابه را دارد؟

با وجود آن‌که کارهای متنوعی در زمینه پیشگوئی خطا ارائه‌شده‌است، اما اولا در برخی موارد با وجود تشابه هدف نهایی تفاوت‌هایی در فرض‌های اولیه وجود دارد. برای نمونه روش ارائه‌شده در [16] بیشتر بر میزان عدم تمرکز برنامه‌نویس استوار است. ثانیا، در بیشتر مواقع پیاده‌سازی کارتحقیقی در دسترس نیست و باز تولید آن می‌تواند شائبه عدم وجود نگاه عادلانه در انجام آزمایش‌ها را تقویت کند. ثالثا برخی ابزارها صرفاً روی یک و فقط یک پروژه کامل قابل اعمال هستند و باید تاریخچه آن پروژه را نیز در دسترس داشته باشند [17]، که این مورد عملا در مورد اکثر مجموعه‌های داده‌ای امکان‌پذیر نیست.

در این مقاله ابزار PMD را جهت رقابت با ابزار پیشنهادی انتخاب می کنیم. این ابزار مبتنی بر مدل بوده و با توجه به مدل‌های محتمل خطایی که در پایگاه یادگیری خود دارد می‌تواند خطادار بودن فایل را تشخیص دهد. در مقابل، برای روش پیشنهادی نیز جهت اعمال روی هرکدام از پروژه‌ها، دیگر پروژه‌ها را به‌عنوان مجموعه یادگیری آن در نظر می‌گیریم.

جدول (۳) نتایج این آزمایش را گزارش می‌دهد. در این جدول G5 همان روش پیشنهادی گروه ۵ (این مقاله) می‌باشد.

در اکثر موارد، برای هر سه معیار precision، recall و accuracy با فاصله محسوسی ابزار پیشنهادی بهتر عمل می‌کند، فقط برای پروژه ecf و tomcat70 صرفاً میزان accuracy ابزار PMD بالاتر نشان می‌دهد.

جدول (۳) نتایج رقابت

	Precision		Recall		Accuracy	
نام پروژه	**PMD**	**G5**	**PMD**	**G5**	**PMD**	**G5**
ecf	۰,۵۳	**۰,۷۴**	۰,۷۰	**۰,۰۶**	**۰,۵۳**	۰,۵۲
eclipse.jdt.core	۰,۲۴	**۰,۶۷**	۰,۸۵	**۰,۳۷**	۰,۴۲	**۰,۸۳**
eclipse.jdt.debug	۰,۴۱	**۰,۷۷**	۰,۷۴	**۰,۲۰**	۰,۵۳	**۰,۶۹**
eclipse.jdt.ui	۰,۳۳	**۰,۸۰**	۰,۷۴	**۰,۳۴**	۰,۴۵	**۰,۷۷**
eclipse.pde.ui	۰,۴۲	**۰,۷**	۰,۷۷	**۰,۳۵**	۰,۵۰	**۰,۶۹**
tomcat70	۰,۶۵	**۰,۹۸**	۰,۷۳	**۰,۰۵**	**۰,۶**	۰,۴۲

۳-۴- جواب سؤال پژوهشی ۲: آیا ابزار ارائه‌شده سرعت قابل قبولی در پیشگوئی دارد؟

این آزمایش روی لپ‌تاپ شخصی با پردازنده Intel® Core i3-4150 3.50 GHz تحت سیستم عامل Windows 7 و در قسمت کنسول محیط Eclipse اجرا شد. مدت زمان پیشگویی هرکدام از پروژه‌ها و متوسط زمان مورد نیاز برای هر فایل هر پروژه در جدول (۴)

گزارش می‌شـود. این جدول نشـان می‌دهد که پیشـگویی هرکدام از ۳۰۰۰۰ فایل کل پروژه‌ها در کسری از ثانیه قابل انجام است.

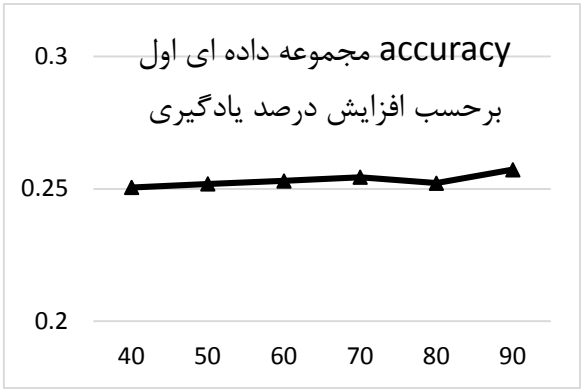
جدول (٤) مدت زمان مورد نیاز برحسب میلی ثانیه		
نام پروژه	مدت زمان پیشگوئی کل پروژه	متوسط زمان پیشگوئی هر فایل در پروژه مربوطه
ecf	۱۷۶۵۲۶	۶۳
eclipse.jdt.core	۶۲۰۳۴۰	۱۰۵
eclipse.jdt.debug	۱۶۵۴۸۴	۱۰۸
eclipse.jdt.ui	۵۷۳۷۴۵	۵۲
eclipse.pde.ui	۳۸۲۷۲۰	۷۱
tomcat70	۲۲۶۶۷۸	۱۲۳

۵-۳- جواب سـؤال پژوهشی ۳: آیا ابزار ارائه‌شده در این مقاله قابلیت اعمال روی پروژه مشابه را دارد؟

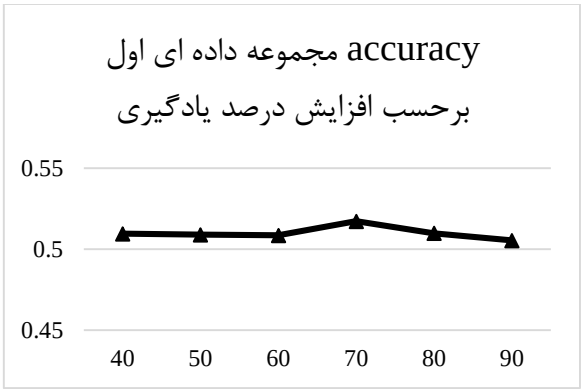
برای پاسخ به این سؤال، ابزار را فقط با یادگیری از یک پروژه برای آن پروژه اعمال می کنیم. برای مثال برای اعمال کردن روی پروژه ecf بخشی از فایل‌های ecf را برای یادگیری و بخشی را برای آزمون در نظر می‌گیریم. این آزمـایش را روی هرکـدام از پروژه‌هـا اعمـال کردیم و همانطور که بیان شد، برای هر پروژه، یادگیری را فقط روی همان پروژه محـدود کردیم، طوری‌کـه ۷۵ درصـد از فـایـل‌های هر پروژه را برای یادگیری و ۲۵ درصـد باقیمانده را برای آزمون تخصـیص دادیم. جدول (۵) نتایج این آزمایش را خلاصه می کند.

جدول (٥) نتایج محدودیت یادگیری ازپروژه مشابه				
ردیف پروژه	نام پروژه	Precision	Recall	Accuracy
۱	ecf	۰،۵۲	۰،۰۸	۰،۶۹
۲	eclipse.jdt.core	۰،۲۵	۰،۹۸	۰،۲۱
۳	eclipse.jdt.debug	۰،۵۹	۰،۹۵	۰،۴۷
۴	eclipse.jdt.ui	۰،۷۹	۰،۳۷	۰،۹۷
۵	eclipse.pde.ui	۰،۷	۰،۲۶	۰،۸۱
۶	tomcat70	۰،۴۵	۰،۱	۰،۹۶

با وجود آن‌که نتایج حاصـل‌شـده تا حد زیادی برای اکثر پروژه‌ها قانع کننده به نظر می رسد، اما مقایسه با نتایج مربوط به G4 در جدول (۳)، افت کاملا محسوسی برمبنای معیارهای یادشده مشاهده می‌شود. برای پروژه ردیف ۲، نتایج اصـلا قابل قبول به نظر نمی‌رسـد، جایی که مـعـیـارprecision پـایین ۰،۳، معیـار recall بـالای ۰،۹ و معیـار accuracy نیز حدود ۰،۲ اسـت. اما قابل ذکر اسـت که این نمی‌تواند شـکسـتی برای روش پیشـنهادی تلقی شـود، بلکه برعکس، این نتایج نشـان می‌دهـد کـه تـا چـه حد بهره‌گیری از تجارب پروژه‌های دیگر می‌تواند در بهبود کارایی پیشـگوئی تاثیرگذار باشـد. بنابراین در واقع مقایسـه نتایج این جدول با نتایج جدول (۳) دلیلی بر موفقیت انگیزه ارائه الگوریتم پیشنهادی و ابزار ارائه‌شده می‌باشد.



(الف)



(ب)

شکل (٤) میزان معیار accuracy با افزایش درصد تعداد فایل‌های یادگیری

در ادامه آزمایش‌های این زیر بخش، در پاسخ به پرسش پژوهشی ۳، آزمایشـی ترتیب دادیم که در آن ابزار پیشـنهادی روی یک پروژه با در نظر گرفتن درصـدی متغیر از تعداد فایل‌های جهت یادگیری اعمال شد، طوری که در آزمایش اول فقط ٪۴۰، در آزمایش بعدی فقط ٪۵۰، درآزمایش بعدی ٪۶۰، درآزمایش بعدی ٪۷۰، درآزمایش بعدی ٪۸۰ و درآزمایش نهایی ٪۹۰ از فایل‌ها جهت یادگیری درنظر گرفته و بقیه آن جهت آزمون مورداسـتفاده قرار گرفت. این آزمایش را روی هر شـش پروژه اعمـال کردیم. در تمـامی موارد برای تمـامی معیـارهـا نتیجه‌ای مشـابه شـکل (۴) گرفتیم، که در آن فقط معیار accuracy برای دو پروژه اول نشـان داده می‌شود. همان‌طور که بیان شد با توجه به این‌که در تمامی آزمایش‌ها نمودار مشـابهی را برای تمامی پروژه‌ها مشـاهده کردیم، از آوردن دیگر نمودارها صـرف نظر می کنیم. در این آزمایش مشاهده شد که با افزایش مجموعه یادگیری، نه تنها افزایش محسوسی ملاحظه نمی‌شـد، بلکه در برخی مواقع، حتی کارایی به صـورت جزئی کاهش پیدا می کرد. مثلا در نمودار (الف) شـکل (۴) کاملا مشـخص

است که با افزایش درصد تعداد فایل یادگیری از ٪۷۰ به ٪۸۰، بجای افزایش accuracy، کاهش آن را مشاهده می‌کنیم.

شاید بتوان دلیل اصلی این مشاهده را کشف کرد. برای توضیح واضح‌تر سناریویی را فرض کنید که هنگام یادگیری، یک فایل خطادار بوده اما به مرکز ثقل سالم بسیار نزدیکتر است، در مقابل فایل سالمی را فرض کنید که به مرکز ثقل فایل‌های خطادار نزدیک‌تر است. با افزایش چنین فایل‌هایی مطمئنا هر الگوریتم پیشگوئی کارایی مورد انتظار را نخواهد داشت. با افزایش چنین مواردی در حین یادگیری، نه‌تنها نوع یادگیری مفید نبوده بلکه همانطور که مشخص است، در برخی موارد باعث افت کیفیت نیز خواهد بود.

به‌هرحال، سوای اینکه دلیل این نوع رفتار چه مواردی می‌تواند باشد، توجه به این نکته حائز اهمیت است که چگونه با مشارکت دادن دیگر پروژه‌ها در یادگیری، توانستیم به مشکل فوق غلبه و به ارقام قابل اعتنایی در افزایش کیفیت پیشگوئی‌ها برسیم.

۴- تهدیدات علیه اعتبار

شاید یکی از موارد عمومی که باید در هر آزمایش تحقیقی مورد لحاظ قرار داد مجموعه داده‌ای باشد. در مورد مسئله پیشگوئی خطا مجموعه‌های داده‌ای کوچکتر به راحتی (بدون نیاز به هیچ پیش پردازش اولیه) در دسترس بودند اما چنین مجموعه‌هایی با تعداد اندک فایل و حجم محدود به هیچ عنوان نمی‌توانستند در اعتبار سنجی الگوریتم پیشنهادی و مقایسه با رقیب محک مناسبی باشند. استفاده از چنین مجموعه‌های داده‌ای می‌توانست یک تهدید جدی در اعتبارسنجی الگوریتم پیشنهادی و ابزار ارائه‌شده باشد. از این روی در اولین قدم پیاده‌سازی این پروژه به آماده‌سازی مجموعه داده‌ای مناسب پرداختیم.

یکی دیگر از موارد عمومی که باید لحاظ کرد انتخاب رقیب است. طبیعتا پیاده‌سازی نامناسب الگوریتم رقیب نمی‌تواند شرایط عادلانه‌ای را در اعتبار سنجی ابزار یا الگوریتم پیشنهادی فراهم آورد. در خصوص برطرف کردن این تهدید، سعی کردیم تا الگوریتمی را جهت رقابت انتخاب کنیم که علاوه بر در دسترس بودن پیاده‌سازی آن، شرایطی مانند سرعت مناسب را نیز داشته باشد. بالطبع گزارش‌هایی مانند "این الگوریتم (رقیب) قابل اعمال روی این مجموعه داده‌ای نبود" به وفور در کارهای تحقیقی یافت می‌شود. جهت اجتناب از این چنین گزارش‌هایی سعی کردیم تا از برخی الگوریتم‌های ناکارآمد واضح صرف نظر کنیم (مثلا فرض کنید اگر بخواهیم با روش بازیابی اطلاعات پایه و تشابه برداری، تشابه بردار فایل ورودی را با تک تک موارد یادگرفته‌شده بررسی کنیم، طبیعی است که این روش پایه بسیار وقت‌گیر و ناکارآمد خواهد بود).

۵- نتیجه‌گیری

در این مقاله پیاده‌سازی روش پیشنهادی برای پیشگوئی خطاهای یک پروژه را توضیح دادیم. علاوه بر توضیح پیاده‌سازی روش پیشنهادی در فاز قبل، آزمایش‌های متعددی را جهت اعتبارسنجی ابزار پیشنهادی انجام دادیم. در این آزمایش‌ها از یک مجموعه مشتمل بر حدود ۳۰،۰۰۰ فایل برنامه جاوا در قالب ۶ پروژه مختلف استفاده کردیم. نتایج آزمایش‌های انجام‌شده نشان می‌دهد علاوه‌بر کارایی ابزار ارائه‌شده برحسب معیارهای precision، recall و accuracy در مقایسه با نرم‌افزار شناخته شده PMD، سرعت این ابزار نیز قابل توجه بوده طوری که هر فایل موجود در مجموعه داده‌ای را می‌تواند در کسری از ثانیه (کمتر از ۰،۲ ثانیه) پیشگوئی کند.

همچنین پیاده‌سازی کلی این ابزار درقالب یک jar فایل ارئه می‌شود. استفاده و تغییر این ابزار نیز برای هرگونه استفاده ممکن در مقاصد صرفاً پژوهشی مجاز می‌باشد.

۶- کارهای آتی

نتایج آزمایش‌های انجام‌شده نشان داد که به هنگام اعمال الگوریتم پیشنهادی با محدودیت یادگیری از فایل‌های دیگر همان پروژه، افزایش تعداد فایل‌ها جهت یادگیری، نه تنها تاثیری در افزایش معیارهای کارایی ندارد، بلکه در برخی موارد حتی باعث کاهش آن معیارها می‌شود، حال آنکه انتظار افزایش کارایی با افزایش تجربه کاملا منطقی می‌نماید. اما واکاوی دلیل این مشاهده می‌تواند تلنگری برای شروع یک کار تحقیقی جدید با نیل به هدف مشاهده مورد منطقی مورد انتظار باشد. البته در همین مقاله نیز آزمایش‌هایی انجام شد و طبق همان مورد می‌توان دلیل اصلی این مشاهده را کشف کرد. برای توضیح واضح‌تر سناریویی را فرض کنید که هنگام یادگیری، یک فایل خطادار بوده اما به مرکز ثقل سالم بسیار نزدیکتر است، در مقابل فایل سالمی را فرض کنید که به مرکز ثقل فایل‌های خطادار نزدیک‌تر است. با افزایش چنین فایل‌هایی مطمئنا هر الگوریتم پیشگوئی کارایی مورد انتظار را نخواهد داشت. ازجمله راه‌حلی که می‌توان ارائه داد، خوشه‌بندی (بدون نظارت) قبل از یادگیری (با نظارت) خواهد بود. در قدم اول می‌توان بردارهای حاصل را با یکی از الگوریتم‌های خوشه‌بندی مانند k-means دو طبقه کرده و به هنگام یادگیری، فقط از دو دسته استفاده کرد (۱) فایل‌های خطاداری که در یک خوشه قرار دارند (۲) فایل‌های سالمی که در یک خوشه قرار دارند.

۸- مراجع

[1] S. Kim, E. J. Whitehead Jr and Y. Zhang, "Classifying Software Changes: Clean or Buggy?," *IEEE Transactions on Software Engineering*, vol. 34, no. 2, pp. 181-196, 2008.

[2] D. D. Nucci, F. Palomba, G. D. Rosa, G. Bavota, R. Oliveto and A. D. Lucia, "A Developer Centered Bug Prediction Model," *IEEE Transactions on Software Engineering*, vol. 44, no. 1, pp. 5-24, 2018.

[3] S. Shivaji, E. J. Whitehead, R. Akella and S. Kim, "Reducing features to improve code change-based bug prediction," *IEEE Transactions on Software Engineering*, vol. 39, pp. 552-569, 2013.

[4] S. Shivaji, E. J. Whitehead Jr, R. Akella and S. Kim, "Reducing features to improve bug prediction," in *2009 IEEE/ACM International Conference on Automated Software Engineering*, 2009.

[5] D. Kim, Y. Tao, S. Kim and A. Zeller, "Where Should We Fix This Bug? A Two-Phase Recommendation Model," *IEEE Transactions on Software Engineering*, vol. 39, no. 11, pp. 1597-1610, 2013.

[6] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013.

[7] M. Rada, "Unsupervised large-vocabulary word sense disambiguation with graph-based algorithms for sequence data labeling," *Proc. HLT. ACL*, 2005.

[8] X. Ye, H. Shen, X. Ma, R. Bunescu and C. Liu, "From word embeddings to document similarities for improved information retrieval in software engineering," in *Proceedings of the 38th international conference on software engineering*, 2016.

[9] S. Brin and L. Page, "The Anatomy of a Large-scale Hypertextual Web Search Engine," *Comput. Netw. ISDN Syst.*, vol. 30, pp. 107-117, 4 1998.

[10] E. Enslen, E. Hill, L. Pollock and K. Vijay-Shanker, "Mining source code to automatically split identifiers for software analysis," 2009.

[11] R. K. Saha, M. Lease, S. Khurshid and D. E. Perry, "Improving Bug Localization Using Structured Information Retrieval," in *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, Piscataway, 2013.

[12] R. Wu, H. Zhang, S.-C. Cheung and S. Kim, "CrashLocator: Locating Crashing Faults Based on Crash Stacks," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, New York, NY, USA, 2014.

[13] X. Ye, R. Bunescu and C. Liu, "Learning to Rank Relevant Files for Bug Reports Using Domain Knowledge," in *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, New York, NY, USA, 2014.

[14] J. Zhou, H. Zhang and D. Lo, "Where Should the Bugs Be Fixed? - More Accurate Information Retrieval-based Bug Localization Based on Bug Reports," in *Proceedings of the 34th International Conference on Software Engineering*, Piscataway, 2012.

[15] M. Nayrolles and A. Hamou-Lhadj, "CLEVER: Combining Code Metrics with Clone Detection for Just-in-time Fault Prevention and Resolution in Large Industrial Projects," in *Proceedings of the 15th International Conference on Mining Software Repositories*, New York, NY, USA, 2018.

[16] D. Di Nucci, F. Palomba, G. De Rosa, G. Bavota, R. Oliveto and A. De Lucia, "A developer centered bug prediction model," *IEEE Transactions on Software Engineering*, vol. 44, pp. 5-24, 2018.

[17] "findbugs," [Online]. Available: http://findbugs.sourceforge.net/.