تشخیص کدهای مشکوک در کد منبع با استفاده از تکنیک های داده کاوی

مینا ریاحی عالم[۱]، رضا اکبری[۲]

[۱] کارشناس ارشد، مهندسی کامپیوتر- نرم افزار، دانشگاه صنعتی شیراز ، شیراز،
m.riahialam@gmail.com

[۲] استادیار، دانشکده مهندسی کامپیوتر و فناوری اطلاعات، دانشگاه صنعتی شیراز، شیراز،
akbari@sutech.ac.ir

چکیده

با توجه به گسترش نرم افزارها، مهندسین نرم افزار و محققین به دنبال کاهش هزینه های تولید نرم افزار و افـزایش کیفیـت آن از جمله قابلیت توسعه، قابلیت پیمانه ای، استفاده مجدد، پیچیدگی، نگهداری و کارایی می باشند. بازسازی نرم افزار یکی از روش های موثر در افزایش کیفیت نرم افزار می باشد. یکی از نکات مورد نظر در بازسازی، تشخیص کدهای مشکوک در نرم افـزار اسـت. کـد های مشکوک به علت ناکارا بودن طراحی یا پیاده‌سازی ناقص می باشند که تکامل، درک و حفظ نرم‌افزار را دشوار می نمایند. از این رو روش هایی جهت تشخیص کدهای مشکوک در نرم افزار مورد مطالعه قرار گرفته اند. یکی از روش های مورد استفاده، الگـوریتم های داده کاوی می باشند که به بهبود صحت در تشخیص کدهای مشکوک کمک می نمایند. اما یکـی از مسـائلی کـه در داده کـاوی مطرح است انتخاب ویژگی مناسب می باشد. روش های مختلفی از جمله الگوریتم های فراابتکاری جهت این امر مناسب می باشند. در این مقاله با استفاده از الگوریتم جستجوی فاخته، ابتدا عمل انتخاب ویژگی انجام شده است، سپس با استفاده از الگـوریتم هـای داده کاوی، الگوریتم ها جهت تشخیص کدهای مشکوک بررسی شده اند.

کلمات کلیدی

کد مشکوک، داده کاوی، انتخاب ویژگی، الگوریتم های فراابتکاری، الگوریتم جستجوی فاخته.

۱- مقدمه

امروزه با افزایش وسعت نرم افزارها، فرآیند توسعه نرم افزار نیز گسترش یافته است. فرآیند توسعه ی نرم افزار می تواند شامل مستند سازی، تست نرم افزار، توسعه ی جدید، نگهداری و استفاده ی مجدد باشد. با توجه به پیشرفت دنیای تکنولوژی امروزه نیاز به نرم افزارهایی با کیفیت بیشتر مـی باشـد. بـه همـین جهت کیفیت نرم افزار با معیارهای گوناگونی مـورد ارزیـابی قـرار مـی گیـرد. همچنین راه های مختلفی جهت افزایش کیفیت نرم افزار ها موجود می باشند. اگر مراحل مختلف توسعه ی نرم افزار به خوبی صورت گیرد، کیفیت افـزایش خواهد یافت. اما برخی مواقع به دلیل خطاهای موجود در مراحل توسعه ی نرم افزار ایراداتی ایجاد می شود که هزینه بر هسـتند. یکـی از راه هـای افـزایش کیفیت نرم افزار بر طرف نمودن عیوب نرم افزار در مراحل مختلف توسعه می باشد. اگر این عیوب برطرف نشوند، باعث به وجـود آمـدن بـدهی فنـی[۱] مـی شود[۱]. پیامد های بدهی فنی بر کیفیت نرم افزار به طور تجربی ثابـت شـده است. به عنوان مثال طراحی ضعیف یا پیاده سازی ضعیف یکی از موارد ایجاد بدهی ضعیف خواهد شد و امکان دارد بـر روی نگهـداری یـک سیسـتم نـرم افزاری اثر بگذارد.

از این رو جهت افزایش کیفیت نرم افزار نیاز است تا اطلاعاتی در ارتباط با نرم افزار کسب نمود. مخازن نرم‌افزار[۲] شامل اطلاعات زیادی پیرامون نرم‌افزار می‌باشند که با استخراج اطلاعات مفید می توان به روند تکامل نرم‌افزار کمک نمود. یکی از مخازن نرم افزار مورد استفاده منبع کد است[۲]. بدین صورت می توان با بدست آوردن متریک های مناسب از منبع کد، به رفع نواقص موجود در پیاده سازی کمک نمود که این کار باز سازی[۳] نرم افزار می باشد. در مفهوم تکامل نرم‌افزار[۴]، بازسازی نیز به بهبود کیفیت نرم‌افزار از جمله قابلیت توسعه[۵]، قابلیت پیمانه ای[۶]، استفاده مجدد، پیچیدگی، نگهداری و کارایی[۷]، به کار می روند[۳]. به این نکته هم اشاره شود که برخی مواقع نیز یک پیاده سازی ناقص ناشی از یک طراحی ضعیف می باشد.

بازسازی در نرم افزار به این صورت است که ابتدا بر روی منبع کد تحلیل صورت می گیرد. سپس بررسی می شود که آیا کد مورد نظر نیاز به بازسازی دارد یا خیر. در واقع در این قسمت کد های مشکوک[۸] تشخیص داده می شوند. در پایان نیز اگر نیاز باشد، بازسازی صورت می پذیرد. نرم افزارها دارای معیارهای مختلفی می باشند که می توان با استفاده از آنها ویژگی های مختلف یک نرم افزار مشخص نمود.

همانطور که بیان شد، منبع کد دارای اطلاعاتی است که می توان به وسیله ی در نظر گرفتن متریک های مناسب به استخراج اطلاعات مفید در جهت بهبود کیفیت نرم افزار پرداخت. بدین منظور نرم افزار هایی برای محاسبه ی متریک ها از کد نرم افزار در دسترس می باشند. به عنوان نمونه می توان به نرم افزار اکلیپس[۹] اشاره نمود. به منظور تشخیص کدهای مشکوک در نرم افزار متریک هایی از کد در نظر گرفته شده اند. از جمله متریک های مورد توجه در نرم افزار می توان به تعداد خطوط کد یک عملگر[۱۰]، تعداد خطوط کد، تعداد بسته[۱۱]، تعداد کل کلاس های در یک سیستم، در یک بسته و یا یک کلاس و غیره اشاره نمود[۴].

در ادامه ابتدا مروری بر تحقیقات پیشین که بر روی کدهای مشکوک و روش های مختلف جهت انتخاب ویژگی پرداخته می شود. سپس در بخش دوم، مفاهیم اولیه ی مورد نیاز در این پژوهش بیان شده است. بخش سوم نیز شامل روش پیشنهادی جهت تشخیص کد مشکوک و نحوه ی پیاده سازی و نتایج بدست آمده می باشد. در پایان نیز نتیجه گیری حاصل از این تحقیق صورت گرفته است.

۱-۱- مروری بر تحقیقات پیشین

با توجه به تاثیرکد مشکوک در نرم افزار، تحقیقات مختلفی پیرامون روش های تشخیص آنها انجام شده است. در[۵] ، مانیرات[۱۲] و همکارانش به پیش‌بینی کدهای مشکوک از جمله کلاس تنبل، خصیصه ی حسادت، تابع طولانی، لیست پارامتر طولانی، دستور سوییچ، زنجیر پیام و مرد واسطه می‌پردازد. به همین منظور از مجموعه داده‌هایی استفاده شده است که به وسیله ی هفت الگوریتم یادگیری ماشین بیز ساده،رگرسیون لجستیک[۱۳]، IB1، IBk، VFI، J48 و جنگل تصادفی به وسیله ی نرم افزار وکا آموزش و تست شده اند. داده های مورد استفاده با توجه به معیارهای مختلف نرم افزار مانند پیچیدگی، اطلاعات پایه، ارث بری و غیره در نظر گرفته شده اند. در نتایج بررسی شده در این تحقیق، مشخص شد که الگوریتم های بیز ساده، VFI و J48 صحت کمتری نسبت به الگوریتم های در نظر گرفته شده جهت پیش بینی کد مشکوک دارند.

یکی از روش های مورد استفاده در تشخیص کد های مشکوک با استفاده از الگوریتم های داده کاوی، [۶] می باشد. در این روش شش الگوریتم یادگیری ماشین موجود در وکا شامل J48، SMO، جنگل تصادفی، JRip و شبکه ی بیزی بر روی مجموعه داده کوالیتاس کراپس[۱۴] به کار برده است. کوالیتاس کارپس یک مجموعه داده ی کاملی می باشد که شامل ۱۱۱ نرم افزار است. همچنین یکی از مزیت های دیگر ای مجموعه داده این است که نسخه های مختلف هر یک از نرم افزار ها را در نظر گرفته است. در این روش داده ها از ۷۴ سیستم نرم افزاری می باشند که از مجموعه داده ی کوالیتاس کراپس گرفته شده اند. طبق پیاده سازی انجام شده جهت تشخیص چهار کد مشکوک، خصیصه حسادت، کلاس داده، کلاس بزرگ و متد بزرگ، J48، SMO، جنگل تصادفی و JRip صحتی بیش از ۹۰ درصد بر روی کل داده ها و به طور متوسط بهترین کارایی را نسبت به الگوریتم شبکه ی بیزی داشته اند.

در [۷] نیز که در ادامه تحقیقات پیشین بر روی الگوریتم های داده کاوی در [۶] می باشد، ۱۶ الگوریتم مختلف یادگیری ماشین از جمله B-J48 هرس شده، B-J48 هرس نشده، B-J48 کاهش خطای هرس، B – درخت تصادفی، B-JRip، B بیز ساده، B- SMO RBF Kernel و B- SMO کرنل چند جمله ای، LibSVM C-SVC کرنل خطی[۱۵]، LibSVM C-SVC کرنل های چند جمله ای، LibSVM C-SVC کرنل شعاعی[۱۶]، LibSVM C-SVC کرنل ساگموید[۱۷]، LibSVM v-SVC کرنل خطی، LibSVM v-SVC کرنل چند جمله ای، LibSVM v-SVC کرنل شعاعی و LibSVM v-SVC کرنل ساگموید به کار رفته است که در این پژوهش می توان به اضافه شدن الگوریتم بوستینگ به الگوریتم های گذشته اشاره نمود که تأثیر به سزایی در افزایش صحت الگوریتم ها دارا بوده است. این پژوهش همانند کار قبل بر روی چهار کد مشکوک، خصیصه حسادت، کلاس داده، کلاس بزرگ و متد بزرگ به وسیله ی نرم افزار وکا اجرا و بر روی ۷۴ سیستم نرم افزاری در مجموعه داده کوالیتاس کراپس در نظر گرفته شده است که الگوریتم ها در اعتبارسنجی ضربدری به بالاترین کارایی دست یافته اند که بالاترین کارایی به وسیله J48 و جنگل تصادفی بدست آمده است و بدترین کارایی توسط SVM کسب شده است. همچنین میزان صحت در تشخیص کدهای مشکوک نیز درنظر گرفته شده است. یکی از ایرادات این روش عدم انتخاب ویژگی بر روی داده های بدست آمده از منبع کد نرم افزارها می باشد. یکی از مسائلی که در این روش در نظر گرفته نشده است، مسئله ی انتخاب ویژگی می باشد.

اما در برخی از تحقیقات صورت گرفته، از الگوریتم های بهینه سازی[۱۸] جهت انتخاب ویژگی استفاده شده است. یکی از الگوریتم های مورد بررسی الگوریتم جستجوی فاخته می باشد که در پژوهش های مختلف جهت انتخاب ویژگی به کار رفته است. در پژوهش [۸]، به استفاده از الگوریتم جستجوی فاخته برای انتخاب ویژگی بر روی چهار نوع مجموعه داده پرداخته است. در این روش با تغییر در تابع برازش الگوریتم به بهبودی آن در روند انتخاب ویژگی کمک نموده است.

در پژوهش های مختلف الگوریتم جستجوی فاخته بر روی مجموعه داده های مختلف به خصوص داده های پزشکی مورد استفاده قرار گرفته

است. به عنوان نمونه در [۹]، از این الگوریتم جهت دسته بندی داده های سرطان استفاده کرده است. به دلیل اینکه بیان ژن[۱۹] دارای ویژگی های زیاد و نمونه های کم می باشد، با توجه به معیارهای ارزیابی محاسبه شده برای این الگوریتم، نشان می دهد که الگوریتم جستجوی فاخته مناسب برای طبقه بندی این نوع داده ها می باشد. همچنین در[۱۰]، الگوریتم جستجوی فاخته جهت طبقه بندی داده های سرطان سینه مورد استفاده قرار گرفته است.

در بررسی های صورت گرفته بر روی داده های مختلف، الگوریتم فاخته عملکرد دارای عملکرد مناسبی می باشد. طبق مشاهدات این الگوریتم برای فضای جستجوی بزرگ و محیط های پویا و در حال رشد مناسب است.

۲- بیان مسئله

همانطور که بیان شد، با توجه به افزایش وسعت نرم‌افزارها، مهندسین نرم‌افزار و محققان به دنبال کاهش هزینه‌های تولید نرم‌افزار و افزایش کیفیت‌های مختلف نرم‌افزار می باشند. یکی از موارد مؤثر در کیفیت نرم افزار، منبع کد است که با توجه به افزایش وسعت نرم فزار، افزایش می یابد. به همین منظور مسئله ای که مطرح می شود این است که چگونه می توان روشی بهینه و کارآمد جهت بهبود منبع کد به کار بست.

کدهای مشکوک یکی از مشکلات مؤثر بر کیفیت نرم‌افزار هستند که به علت ناکارا بودن طراحی یا پیاده‌سازی ناقص می باشد[۸]. از این رو مسئله ی شناسایی کدهای مشکوک مهم می باشد. هر کد مشکوک به بررسی نوع خاصی از عناصر سیستم (به عنوان مثال، کلاس و یا متد) می پردازدکه توسط ویژگی های آن بررسی شده است در این پژوهش، به بررسی روشی بهینه جهت تشخیص کدهای مشکوک از منبع کد پرداخته خواهد شد.

یکی از روش های تشخیص کدهای مشکوک، استفاده از الگوریتم های داده کاوی می باشد. یکی از چالش های مهم در داده کاوی و یادگیری ماشین، مسئله ی انتخاب ویژگی می باشد. در واقع انتخاب ویژگی فرآیندی است که در آن زیرمجموعه ای از داده های مرتبط، از داده های اصلی انتخاب می شوند[۱۱]. این کار سبب بهبود کارایی یادگیری، کاهش هزینه محاسبات و بهبود فهم مدل می شود.

۲-۱- کد مشکوک

در بازیابی نرم افزار یکی از نکات اصلی، تشخیص قسمت هایی از کد است که دارای ضعف در طراحی یا پیاده سازی است و نیازمند تغییرات می باشند. طبق تعریف مارتین فولر یک کد مشکوک به معنای یک طراحی ضعیف است[۱۲]. کد مشکوک مانند یک خطای نحوی[۲۰] یا اخطارهای کامپایلر[۲۱] نمی باشد. در واقع به صورت اعلام یک مشکل در طراحی یا برنامه نویسی به صورت عملی است که ممکن است در آینده مسئله ایجاد نماید. تشخیص کد مشکوک به صورت دستی در پروژه های نرم افزاری بزرگ، زمان بر می باشد. از این گذشته این کار هزینه بر و مستعد خطا است. زیرا به تجربه ی توسعه دهنده و میزان درک آن از برنامه وابسته است. در[۱۳]، مانتیلا[۲۲] و همکارانش به دسته بندی کدهای مشکوک با توجه به اشتراکاتی که میان آن ها می باشد، پرداخته اند. از جمله کدهای مشکوک مورد بررسی در این مقاله می توان به کلاس داده، خصیصه ی حسادت، کلاس بزرگ و تابع طولانی اشاره نمود.

- کلاس داده دارای فیلدهایی است که متد های get و set برای این فیلدها می باشند و چیز دیگری ندارند. این قبیل کلاس های نگهدارنده های داده هستند و معمولا توسط کلاس های دیگر مورد دستکاری قرار می گیرند. اگر در اوایل مراحل کار کلاس ها دارای فیلدهای عمومی باشند، می توان به وسیله ی به کارگیری سریع کپسوله سازی فیلد[۲۳] آن را صحیح نمود.
- خصیصه حسادت زمانی رخ می دهد که یک متد به جای کلاس خود، بیشتر به کلاس دیگری تمایل دارد. اغلب تمرکز خصیصه حسادت بر روی داده می باشد. این قسمت را می توان با جابه جایی کد انجام داد.
- تابع طولانی زمانی رخ می دهد که متدی طولانی باشد. برنامه هایی که دارای متد های کوتاه تری هستند عمر بهتر و بیشتری دارند. متدهایی که طولانی می باشند، سخت تر قابل فهم هستند. در زبان های برنامه نویسی قدیمی تر فراخوانی های زیرروال[۲۴]دارای سربار بودند. به همین دلیل برنامه نویسان از استفاده ی متدهای کوچک صرف نظر می کردند. اما زبان های شیء گرایی جدید سربار فراخوانی در فرآیند را تا حدی حذف نموده اند.
- کلاس بزرگ بدین صورت است که زمانی که یک کلاس سعی می کند کارهای زیادی انجام دهد، اغلب دارای متغیرهای زیادی می باشد. زمانی که تعداد متغیرها زیاد باشد، کدهای تکرار شونده اجتناب ناپذیرند و به همین ترتیب کلاس بزرگ می شود.

۲-۲- داده کاوی

با استفاده از الگوریتم های داده کاوی می توان ابتدا به آموزش نمونه های کد مشکوک موجود که برای تشخیص از منبع کد مورد نظر می باشند، استفاده نمود. از جمله‌ی این الگوریتم ها می توان به موارد زیر اشاره نمود.

- درخت تصمیم که یک الگوریتم با ناظر[۲۵] جهت دسته بندی می باشد. درخت تصمیم از ریشه[۲۶] آغاز می شود، سپس به شاخه[۲۷] ها که شامل گره[۲۸] ها می باشند، می شود و در نهایت تصمیم گیری در برگ[۲۹] صورت می گیرد[۱۴]. یکی از الگوریتم های مورد استفاده C4.5 می باشد که یک پیاده سازی کمی تغییر یافته از الگوریتم ID3 است. این الگوریتم با هرس درخت به بالا بردن صحت یادگیری کمک می نماید و باعث کاهش شاخه های تکراری می شود. از جمله الگوریتم های مورد استفاده می توان به J48 و جنگل تصادفی[۳۰] اشاره نمود.
- ماشین بردار پشتیبان[۳۱] یا SVM یکی از روش های یادگیری با نظارت است که از طبقه بندی و رگرسیون[۳۲] استفاده می نماید.
- JRip، یک یادگیرنده مبتنی بر استنتاج و مبتنی بر قواعد (RIPPER[۳۳])است که تلاش می کند قوانین گزاره ای را که می تواند برای طبقه بندی ویژگی ها استفاده شود، ارائه می دهد[۱۵].

- بیز ساده[۳۴]یکی از روش های دسته بند کننده ی با ناظر است. بر مبنای تئوری بیز است. در این روش با در نظر گرفتن احتمال های شرطی موجود در بین ویژگی عمل می نماید.
- آدابوست[۳۵]از دسته ی الگوریتم های بوستینگ[۳۶] است[۱۶]. بوستینگ یک الگوریتم آموزشی است که جهت افزایش کارایی الگوریتم های یادگیری ماشین مورد استفاده قرار می گیرد. آدابوست برای داده های نامتوان[۳۷] و پرت به کار گرفته می شوند و نتایج مربوط به این دسته از داده ها را بهبود می بخشند. این الگوریتم با کلاسه بندی داده ها به آموزش می پردازد.

۲-۳- الگوریتم های بهینه سازی

الگوریتم های بهینه سازی یا جستجوی محلی به دنبال یافتن پاسخی بهینه از مجموعه ی فضای جستجو می باشند [۱۷]. این الگوریتم ها به دو دسته ی الگوریتم های دقیق[۳۸] و الگوریتم‌های تقریبی[۳۹] تقسیم می شوند. اما الگوریتم های تقریبی می توانند جواب‌های خوب و نزدیک به بهینه را در زمان حل کوتاه بیابند. این الگوریتم ها برای مسائل بهینه‌سازی سخت، مناسب می باشند. یکی از الگوریتم های تقریبی، الگوریتم فراابتکاری می باشد[۱۸]. دو جزء اصلی در هر الگوریتم فراابتکاری، انتخاب بهترین راه حل و تصادفی بودن آن است. الگوریتم را می توان انتخاب بهترین جواب اشاره به نزدیک بودن راه حل ها به جواب مطلوب می باشد. همچنین تصادفی بودن آن نشان دهنده ی عدم به دام افتادن در نقطه ی بهینه محلی است. در این تحقیق نیز از این دسته از الگوریتم ها استفاده شده است. الگوریتم جستجوی فاخته از الگوریتم های فراابتکاری می باشد. الگوریتم جستجوی فاخته یا CS روش بهینه‌سازی فراابتکاری است که رویکردی تکاملی در جستجوی راه‌حل بهینه دارد. این الگوریتم در سال ۲۰۰۹ توسط زین-شی یانگ[۴۰] و سوآش دب [۴۱] ،الهام گرفته شده از رفتار پرنده ای به نام فاخته می باشد، توسعه یافته است[۱۹]. این الگوریتم اخیرا نشان داده است که از الگوریتم های ژنتیک و این بهینه سازی جمعیت ذرات[۴۲] مؤثر تر عمل کرده است. پرندگان فاخته به این صورت عمل می کنند که تخم های خود را در لانه های عمومی قرار می دهند. در این بین ممکن است حتی تخم های دیگر را به دلیل افزایش احتمال تبدیل تخم های خود به جوجه، از بین ببرند. تعداد زیادی از گونه ها، در لانه های میزبان تخم گذاری می کنند.

به طور کلی الگوریتم جستجوی فاخته به صورت زیر عمل می نماید:

۱- هر فاخته یک تخم یک مرتبه می گذارد و آن را در یک لانه ی تصادفی رها می کند.
۲- بهترین لانه ها با بالاترین کیفیت تخم به نسل های بعدی منتقل خواهد شد.
۳- تعداد لانه های میزبان موجود ثابت می باشد و تخم گذاشته شده توسط فاخته با احتمال $p_a \geq (۰،۱)$ توسط فاخته ی میزبان کشف می شود. در این مورد، پرنده ی میزبان می تواند تخم را بیرون کند یا به سادگی لانه را ترک کند و یک لانه ی کاملا جدید ایجاد نماید.

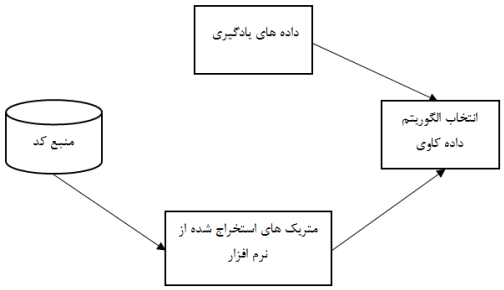
جهت حل مسئله با الگوریتم فاخته می توان هر تخم در یک لانه نمایانگر یک راه حل است و هر فاخته فقط می تواند یک تخم بگذارد. هدف آن است تا از راه حل‌های بهتر و جدید بالقوه استفاده شود تا به جای راه حل هایی که زیاد ایده آل نمی باشند، در لانه ها جایگزین شود. در ساده‌ترین حالت، هر لانه یک تخم دارد. الگوریتم این توانایی را دارد که برای مسائل پیچیده تر بسط داده شود.

دلیل اینکه این الگوریتم عملکرد بهتری نسبت به برخی الگوریتم های دیگر دارد این است که دارای دو قابلیت جستجو می باشد. جستجوی محلی در ۱/۴ زمان جستجو و جستجوی عمومی[۴۳] در ۳/۴ زمان جستجو انجام می شود. این نشان دهنده ی آن است که در فضای عمومی بیشتر عمل کاوش صورت می پذیرد. این حالت زمانی رخ می دهد که P_a =۰،۲۵ در نظر گرفته شود.

۲-۴- تشخیص کد با استفاده از تکنیک های داده کاوی

به طور کلی روش تشخیص کد مشکوک با استفاده از الگوریتم های یادگیری ماشین از منبع کد را می توان به سه مرحله تقسیم نمود،شکل(۱):

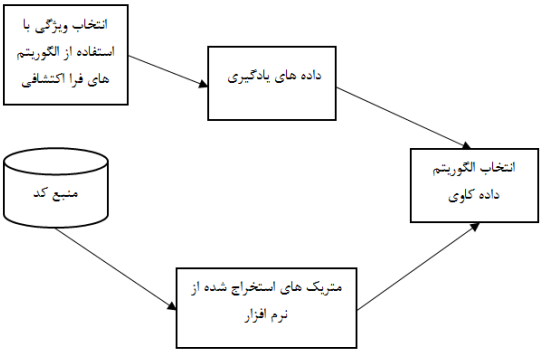
۱- انتخاب داده های یادگیری مناسب که جهت آموزش به الگوریتم داده کاوی می باشند.
۲- آنالیز منبع کد و استخراج ویژگی های مناسب
۳- انتخاب الگوریتم مناسب با توجه به کد مشکوک مورد نظر.



شکل (۱) : روش کلی تشخیص کد مشکوک با استفاده از الگوریتم های داده کاوی

۳- روش پیشنهادی تشخیص کد مشکوک

در روش های تشخیص مبتنی بر داده کاوی، جهت انتخاب، روشی مناسب تر است که بتوان بهترین زیر مجموعه را انتخاب و ایجاد نماید. از جمله راه هایی که می توان انتخاب ویژگی مناسبی از مجموعه ویژگی های داده انجام داد، روش های فراابتکاری می باشند. روش های فراابتکاری به دنبال یافتن بهینه ترین نقطه در فضای جستجو می باشد. از جمله الگوریتم مورد بررسی، الگوریتم جستجوی فاخته می باشد. الگوریتم جستجوی فاخته به دلیل سرعت، همگرایی مؤثر، پیچیدگی کمتر در پیاده سازی و پارامترهای ورودی یکی از الگوریتم های مناسب جهت انتخاب ویژگی استفاده نمود.



شکل (۲) : روش پیشنهادی تشخیص کد مشکوک با استفاده از الگوریتم های داده کاوی

۳-۱- پیاده سازی روش پیشنهادی و نتایج

نحوه ی پیاده سازی به این صورت است که داده های مورد استفاده در این تحقیق با توجه به پژوهش [۷]، در نظر گرفته شده اند. داده ها از مجموعه ی کوالیتاس کراپس با استفاده نرم افزارهای استخراج متریک از نرم افزار ایجاد شده اند. همچنین این مجموعه داده با استفاده از سه نرم افزار مورد تحلیل قرار گرفته شده است تا کدهای مشکوک در آن ها تشخیص داده شود. سپس به صورت دستی داده ها چک شده اند که آیا تشخیص درست بوده است یا خیر. این مجموعه داده ها به چهار دسته ی کلاس داده، خصیصه حسادت، کلاس بزرگ و تابع طولانی تقسیم می شوند.

هر یک از داده ها مطابق و متریک های تعریف شده، دارای ویژگی هایی می باشند. دو مجموعه داده ی کلاس داده و کلاس بزرگ دارای ۶۶ ویژگی و ۴۲۰ نمونه و خصیصه حسادت و تابع طولانی دارای ۸۸ ویژگی و ۴۲۰ نمونه می باشند. روش مطالعه شده که مورد ارزیابی با روش پیشنهادی قرار خواهد گرفت، از۱۶ الگوریتم یادگیری ماشین به تنهایی استفاده نموده است که نتایج نشان دهنده ی آن است که الگوریتم j48 و جنگلی تصادفی بالاترین میزان کارایی را دارا می باشند. این پژوهش بر روی الگوریتم هایی که صحت بالاتری نسبت به بقیه ی الگوریتم ها داشته اند که شامل الگوریتم B-J48 هرس شده، B-J48 هرس نشده، B-J48 کاهش خطای هرس، B – درخت تصادفی و J48 پیاده سازی شده اند. در این پژوهش نیز جهت کاهش ابعاد مجموعه ی داده از انتخاب ویژگی استفاده ننموده است.

ابزار به کار رفته جهت پیاده سازی الگوریتم ها نرم افزار وکا نسخه ی ۳,۸,۱ است. ابتدا توسط الگوریتم جستجوی فاخته انتخاب ویژگی صورت می گیرد که تعداد تکرار جهت پیاده سازی جستجوی فاخته ۲۰ بار در نظر گرفته شده است. در جدول (۱)، (۲)، (۳) و (۴) به ترتیب نتایج صحت پیاده سازی روش پیشنهادی بر روی داده های کلاس داده، خصیصه ی حسادت، کلاس بزرگ و تابع طولانی که با الگوریتم جستجوی فاخته، ابتدا انتخاب ویژگی شده است، سپس با الگوریتم داده کاوی در نظر گرفته شده، نشان داده اند. در داده های کلاس داده، B- j48 هرس نشده و B–جنگل تصادفی بهترین صحت را نسبت به سایر الگوریتم ها با انتخاب ویژگی را دارا می باشند. در انتخاب ویژگی با الگوریتم جستجوی فاخته بر روی داده های خصیصه حسادت B-Jrip بهترین صحت را نسبت به سایر الگوریتم ها در همان انتخاب ویژگی را دارا می باشند. در انتخاب ویژگی با الگوریتم جستجوی فاخته B– جنگل تصادفی بهترین صحت را نسبت به سایر الگوریتم ها در همان انتخاب ویژگی را دارا می باشند. در داده های تابع طولانی، B– جنگل تصادفی بهترین صحت را داراست.

بدین ترتیب الگوریتم B–جنگل تصادفی یکی از بهترین عملکردها را نسبت به سایر الگوریتم ها با انتخاب ویژگی توسط الگوریتم جستجوی فاخته داشته است.

۴- نتیجه گیری

همانطور که بیان شد، مسئله ی بازسازی کد نرم افزار یکی از مواردی است که به افزایش کیفیت نرم افزار کمک می نماید. جهت بازسازی کد نرم افزار نیاز است تا ابتدا نقاطی از کد نرم افزار که نیاز به بازسازی دارند، تشخیص داده شوند. طبق بررسی های صورت گرفته در این پژوهش، با استفاده از الگوریتم جستجوی فاخته و تکنیک های مختلف داده کاوی مشخص شد که الگوریتم B–جنگل تصادفی با الگوریتم جستجوی فاخته دارای یکی از بهترین عملکرد نسبت به روش دیگر، در مسئله ی تشخیص کدهای مشکوک را داراست.

همچنین با توجه به بررسی ویژگی های انتخاب شده توسط الگوریتم جستجوی فاخته مشخص شد که این الگوریتم با انتخاب ویژگی های مربوط به همان کد مشکوک زیرمجموعه ای مناسب را انتخاب می نمایند.

جدول (۱) صحت پیاده سازی با داده های کلاس داده

		بدون انتخاب(مورد مطالعه)	الگوریتم انتخاب ویژگی جستجوی فاخته
الگوریتم های داده کاوی	B- j 48 هرس شده	۹۹,۰۲	۹۸,۰۴
	B- j 48 هرس نشده	۹۸,۶۷	۹۸,۷۲
	B- j 48 کاهش خطای هرس	۹۸,۰۷	۹۸,۰۲
	B- Jrip	۹۸,۸۳	۹۸,۱
	B جنگل تصادفی	۹۸,۵۷	۹۸,۷۲
	B بیز ساده	۹۷,۳۳	۸۳,۴۷
	B- SMO RBF Kernel	۹۷۰۱۴	۹۰,۰۵
	B- SMO کرنل های چند جمله ای	۹۶,۹	۹۴,۹۳
	j 48	۹۸,۵۵	۸۹,۷۶
	Jrip	۹۸,۱۷	۹۷,۰۲

جدول (۲) صحت پیاده سازی با داده های خصیصه حسادت

		بدون انتخاب(مورد مطالعه)	الگوریتم انتخاب ویژگی جستجوی فاخته
الگوریتم های داده کاوی	B- j 48 هرس شده	۹۶,۶۲	۹۶,۳۱
	B- j 48 هرس نشده	۹۶,۵	۹۶,۲۹
	B- j 48 کاهش خطای هرس	۹۵,۹	۹۵,۷۴
	B- Jrip	۹۶,۶۴	۹۶,۸
	B جنگل تصادفی	۹۶,۴	۹۶,۷۹
	B بیز ساده	۹۱,۵	۸۴,۳
	B- SMO RBF Kernel	۹۳,۸۸	۸۹,۴۸
	B- SMO کرنل های چند جمله ای	۹۲,۰۵	۹۱,۰۵
	j 48	۹۵,۹۵	۹۶
	Jrip	۹۵,۶۷	۹۵,۹۷

جدول (۳) صحت پیاده سازی با داده های کلاس بزرگ

		بدون انتخاب(مورد مطالعه)	الگوریتم انتخاب ویژگی جستجوی فاخته
الگوریتم های داده کاوی	B- j 48 هرس شده	۹۷,۰۲	۹۶,۹۸
	B- j 48 هرس نشده	۹۷,۰۲	۹۶,۹۵
	B- j 48 کاهش خطای هرس	۹۷,۲۶	۹۷,۱۲
	B- Jrip	۹۶,۹	۹۶,۸۸
	B جنگل تصادفی	۹۶,۹۵	۹۷,۳۳
	B بیز ساده	۹۷,۵۴	۹۴,۲۱
	B- SMO RBF Kernel	۹۴,۶۲	۹۵,۱۴
	B- SMO کرنل های چند جمله ای	۹۴,۳۳	۹۶,۳۶
	j 48	۹۷,۳۱	۹۶,۷۹
	Jrip	۹۷,۱۲	۹۷,۱۲

جدول (۳) صحت پیاده سازی با داده های تابع طولانی

		بدون انتخاب(مورد مطالعه)	الگوریتم انتخاب ویژگی جستجوی فاخته
الگوریتم های داده کاوی	B- j 48 هرس شده	۹۹,۴۳	۹۹,۵
	B- j 48 هرس نشده	۹۹,۲	۹۹,۶
	B- j 48 کاهش خطای هرس	۹۹,۱۹	۹۹,۰۷
	B- Jrip	۹۹,۰۳	۹۹,۱۶
	B جنگل تصادفی	۹۹,۲۳	۹۹,۵۵
	B بیز ساده	۹۷,۸۶	۹۷,۱۶
	B- SMO RBF Kernel	۹۷	۹۷,۵۳
	B- SMO کرنل های چند جمله ای	۹۸,۶۷	۹۷,۲۹
	j 48	۹۹,۱	۹۹,۲۷
	Jrip	۹۹,۰۲	۹۹,۱

مراجع

[1] M. Tufano *et al.*, “When and Why Your Code Starts to Smell Bad,” 2015, pp. 403–414.

[2] A. E. Hassan, “The road ahead for Mining Software Repositories,” 2008, pp. 48–57.

[3] T. Mens and T. Tourwe, “A survey of software refactoring,” *IEEE Trans. Softw. Eng.*, vol. 30, no. 2, pp. 126–139, Feb. 2004.

[4] R. S. Pressman, *Software engineering: a practitioner’s approach*, Eighth edition. New York, NY: McGraw-Hill Education, 2015.

[5] N. Maneerat and P. Muenchaisri, “Bad-smell prediction from software design model using machine learning techniques,” 2011, pp. 331–336.

[6] F. A. Fontana, M. Zanoni, A. Marino, and M. V. Mantyla, “Code Smell Detection: Towards a Machine Learning-Based Approach,” 2013, pp. 396–399.

[7] C. C. Aggarwal, Ed., *Data classification: algorithms and applications*. Boca Raton: CRC Press, Taylor & Francis Group, 2014..

[8] L. A. M. Pereira *et al.*, “A Binary Cuckoo Search and Its Application for Feature Selection,” in *Cuckoo Search and Firefly Algorithm*, vol. 516, X.-S. Yang, Ed. Cham: Springer International Publishing, 2014, pp. 141–154.

[9] C. Gunavathi and K. Premalatha, “Cuckoo search optimisation for feature selection in cancer classification:

a new approach," *Int. J. Data Min. Bioinforma.*, vol. 13, no. 3, p. 248, 2015.

[10] M. N. Sudha and S. Selvarajan, "Feature Selection Based on Enhanced Cuckoo Search for Breast Cancer Classification in Mammogram Image," *Circuits Syst.*, vol. 07, no. 04, pp. 327–338, 2016.

[11] Huan Liu and Lei Yu, "Toward integrating feature selection algorithms for classification and clustering," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 4, pp. 491–502, Apr. 2005.

[12] M. Fowler and K. Beck, *Refactoring: improving the design of existing code*. Reading, MA: Addison-Wesley, 1999.

[13] M. Mantyla, J. Vanhanen, and C. Lassenius, "A taxonomy and an initial empirical study of bad smells in code," 2003, pp. 381–384.

[14] Y. Zhao and Y. Zhang, "Comparison of decision tree methods for finding active objects," *Adv. Space Res.*, vol. 41, no. 12, pp. 1955–1959, Jan. 2008.

[15] [12] V. Rajeswari and K. Arunesh, "Analysing Soil Data using Data Mining Classification Techniques," *Indian J. Sci. Technol.*, vol. 9, no. 19, May 2016.

[16] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera, "A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches," *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.*, vol. 42, no. 4, pp. 463–484, Jul. 2012.

[17] H. H. Hoos and T. Stützle, *Stochastic local search: foundations and applications*. San Francisco, CA: Morgan Kaufmann Publishers, 2005.

[18] X.-S. Yang, Engineering optimization an introduction with metaheuristic applications. Hoboken, N.J: John Wiley, 2010.

[19] I. Fister, X.-S. Yang, D. Fister, and I. Fister, "Cuckoo Search: A Brief Literature Review," in *Cuckoo Search and Firefly Algorithm*, vol. 516, X.-S. Yang, Ed. Cham: Springer International Publishing, 2014, pp. 49–62.

پانویس ها

[1] Technical Debt
[2] Software Repositories
[3] Refactoring
[4] Software Evolution
[5] Extensibility
[6] Modularity
[7] Efficiency
[8] Code Smell
[9] Eclipse
[10] Operator
[11] Package
[12] Maneerat
[13] Logistic regression
[14] Qualitas Corpus
[15] Linear Kernel
[16] Ratial Kernel
[17] Sigmoid Kernel
[18] Optimization Algorithm
[19] Gene Expression
[20] Syntax
[21] Compiler
[22] Mäntylä
[23] Encapsulation
[24] Procedure
[25] Supervised
[26] Root
[27] Branch
[28] Node
[29] Leaf
[30] Random Forest
[31] Sector Vector Machine
[32] Regression
[33] Repeated incremental pruning to produce error reduction
[34] Naive Bayes
[35] Adaboost
[36] Boosting
[37] Imbalance
[38] Exact
[39] Approximate
[40] Xin-She Yang
[41] Suash Deb
42 Particle swarm optimization
[43] Global Search