Journal of Electronical & Cyber Defence
Vol. 10, No. 1, Spring 2022, Serial No. 37

Feature selection using a combination of Genetic-Whale-Ant colony algorithms for software fault prediction by machine learning

A. Karimi*, M.R Irajimoghaddam

* Assistant Professor, Imam Hossein University, Tehran, Iran

(Received: 02/05/2021, Accepted: 27/08/2021)

ABSTRACT

Software fault prediction methods are used to predict fault-prone modules in the early stages of software development. Machine learning techniques are the most common techniques used in software fault prediction. Data dimensionality is one of the problems that affect the performance of machine learning algorithms. Data dimensionality means the existence of irrelevant or redundant features that may mislead the learning algorithm hence decrease its accuracy. The low accuracy of software fault prediction causes the late detection of some faulty modules and as a result increases the effort and cost of fixing faults abnormally. Therefore, solving the data dimensionality problem is necessary to increase the accuracy of software fault prediction. Researchers use the feature selection algorithms for dimensionality reduction. Feature selection algorithms are divided into two types of filter-based feature selection and wrapper-based feature selection algorithms. Wrapper-based algorithms lead to higher accuracy prediction models. In these algorithms we can use different methods to search for the good solutions; the best of which, is the metaheuristic search. Each of the metaheuristic algorithms has some strengths and weaknesses, so the researchers use a combination of these algorithms to address these weaknesses. In this research, to address the weaknesses of each metaheuristic algorithm, a combination of genetic, ant colony and whale optimization algorithms, is used as the wrapper feature selection. Obviously, the application of early software fault prediction methods before the actual test is one of the effective passive defense techniques in reducing the software system development costs. 19 software projects are used to evaluate the proposed method. Comparison of the results with other methods shows that the proposed method outperforms the counterparts.

Keywords: Software Fault Prediction, Feature Selection, Metaheuristic Algorithm, Genetic Algorithm, Whale Optimization Algorithm, Ant Colony Optimization Algorithm.

* Corresponding Author Email: a.karimi@ihu.ac.ir

نشریه علمی "پدافند الکترونیکی و سایبری"
سال دهم، شماره ۱، بهار ۱۴۰۱، ص ۴۵-۳۳

علمی-پژوهشی

انتخاب ویژگی با استفاده از ترکیب الگوریتم‌های ژنتیک–وال–کلونی مورچگان، برای پیش‌بینی خطاهای نرم‌افزار مبتنی بر یادگیری ماشین

علی کریمی[1*]، محمدرضا ایرجی‌مقدم[2]، اسماعیل بسطامی[3]

۱- استادیار، ۲- دانشجوی کارشناسی ارشد، ۳- پژوهشگر، دانشگاه جامع امام حسین(ع)، تهران، ایران

(دریافت: ۱۴۰۰/۰۲/۱۲، پذیرش: ۱۴۰۰/۰۶/۰۵)

چکیده

روش‌های پیش‌بینی خطای نرم‌افزار برای پیش‌بینی ماژول‌های مستعد خطا در مراحل اولیه‌ی توسعه‌ی نرم‌افزار استفاده می‌شود. امروزه فنون یادگیری ماشین پرکاربردترین فنون مورد استفاده در زمینه‌ی پیش‌بینی خطاهای نرم‌افزار محسوب می‌شود. ابعاد بالای داده یکی از مشکلاتی است که عملکرد الگوریتم‌های یادگیری ماشین را تحت تأثیر قرار می‌دهد. ابعاد بالای داده به معنای وجود ویژگی‌های غیرمفید یا افزونه است که احتمالاً الگوریتم یادگیری را گمراه می‌کند و درنتیجه دقت آن را کاهش می‌دهد. دقت پایین پیش‌بینی خطای نرم‌افزار باعث شناسایی دیرهنگام بعضی ماژول‌های خطادار می‌شود و در نتیجه تلاش و هزینه‌ی برطرف کردن خطاها را به صورت غیرعادی بالا می‌برد. بنابراین حل مشکل ابعاد بالای داده برای افزایش دقت پیش‌بینی خطای نرم‌افزار ضروری است. برای کاهش ابعاد داده، محققین از الگوریتم‌های انتخاب ویژگی استفاده می‌کنند. الگوریتم‌های انتخاب ویژگی به دو دسته‌ی الگوریتم‌های مبتنی بر فیلتر و الگوریتم‌های مبتنی بر پوشش تقسیم می‌شود. الگوریتم‌های مبتنی بر پوشش منجر به مدل‌های پیش‌بینی با دقت بالاتری می‌شود. در این الگوریتم‌ها می‌توان از روش‌های مختلفی برای جستوجوی راه‌حل‌ها استفاده نمود که بهترین نوع آن جست‌وجوی فراابتکاری است. هرکدام از الگوریتم‌های فراابتکاری نقاط قوت و ضعفی دارد که محققان برای برطرف کردن این نقاط ضعف از ترکیب این الگوریتم‌ها استفاده می‌کنند. در این تحقیق برای بهبود نقاط ضعف هرکدام از الگوریتم‌های فراابتکاری، از ترکیب سه الگوریتم ژنتیک، کلونی مورچگان و بهینه‌سازی وال برای انتخاب ویژگی مبتنی بر پوشش استفاده می‌شود. بدیهی است به‌کارگیری روش‌های پیش‌بینی زودهنگام خطاهای نرم‌افزار قبل از آزمون واقعی آن، یکی از فنون مؤثر پدافند غیرعامل در کاهش هزینه‌های توسعه‌ی سامانه‌های نرم‌افزاری محسوب می‌شود. برای ارزیابی روش پیشنهادی، ۱۹ پروژه‌ی نرم‌افزاری مورد بررسی و آزمایش قرار گرفته و نتایج با دیگر روش‌ها مقایسه شده است. نتایج ارزیابی نشان می‌دهد که روش پیشنهادی از عملکرد بهتری نسبت به سایر روش‌ها برخوردار است.

کلیدواژه‌ها: پیش‌بینی خطای نرم‌افزار، انتخاب ویژگی، الگوریتم فراابتکاری، الگوریتم ژنتیک، الگوریتم بهینه‌سازی وال، الگوریتم کلونی مورچگان

۱- مقدمه

آزمون نرم‌افزار، فنی است که هدف آن بررسی میزان کیفیت نرم‌افزار است. آزمون نرم‌افزار هزینه‌بر است؛ اما ثابت شده است که اجتناب از آن پرهزینه‌تر است [۱]. پیش‌بینی خطای نرم‌افزار[1] بخش مهمی از مهندسی نرم‌افزار و به معنای شناسایی ماژول‌های مستعد خطا در مراحل اولیه‌ی توسعه‌ی نرم‌افزار است. شناسایی خطا در مراحل دیرتر هزینه را به‌صورت غیرعادی بالا می‌برد. مدل‌های پیش‌بینی خطای نرم‌افزار به تیم آزمون کمک می‌کند که بیشتر روی ماژول‌های مستعد خطا تمرکز کنند و بنابراین تلاش و هزینه‌ی لازم را برای آزمون نرم‌افزار کاهش می‌دهد [۲].

امروزه فنون یادگیری ماشین پرکاربردترین فنون مورد استفاده در زمینه‌ی پیش‌بینی خطای نرم‌افزار محسوب می‌شود [۲]. یکی از مشکلات اصلی که کارایی الگوریتم‌های یادگیری را تحت تأثیر قرار می‌دهد، ابعاد بالای داده[2] است. داده با ابعاد بالا ممکن است حاوی ویژگی‌های[3] غیرمفید یا افزونه باشد که احتمال دارد الگوریتم یادگیری را گمراه کند و درنتیجه دقت آن را کاهش دهد. حذف چنین ویژگی‌هایی به افزایش دقت الگوریتم‌های یادگیری ماشین کمک می‌کند. یک راه‌حل ممکن برای کاهش ابعاد داده، انتخاب ویژگی[4] است که هدف آن کاهش ابعاد داده به‌وسیله‌ی حذف داده‌های غیرمفید و افزونه است [۳].

* رایانامه نویسنده مسئول: a.karimi@ihu.ac.ir

[1] Software Fault Prediction

[2] Data Dimesionality
[3] Feature
[4] Feature Selection

ناشر: دانشگاه جامع امام حسین (ع)

مشکل دیگری که در زمینه‌ی پیش‌بینی خطای نرم‌افزار وجود دارد، نبود تعادل[1] در اکثر مجموعه‌داده‌های پیش‌بینی خطای نرم‌افزار است که می‌تواند منجر به دقت پایین پیش‌بینی خطا شود. برای حل این مشکل محققان از روش‌های نمونه‌برداری[2] استفاده می‌نمایند [۴].

رویکردهای متعددی از تعداد ثابتی از ویژگی‌ها یا تمام ویژگی‌های موجود برای پیش‌بینی خطاهای نرم‌افزار استفاده می‌کند. اما استفاده از تمام ویژگی‌ها یا تعداد ثابتی از آن‌ها برای تمام مجموعه‌داده‌ها منجر به یک مدل پیش‌بینی خطا با دقت بالا نمی‌شود [۳]. یعنی تعدادی ویژگی خاص وجود ندارد که با استفاده از آن‌ها بتوان روی هر مجموعه‌داده‌ای، یک مدل پیش‌بینی خطا با دقت بالا ساخت و ویژگی‌های بهینه برای هر مجموعه‌داده ممکن است متفاوت باشند. با در نظر گرفتن روش جست‌وجو، سه راهبرد اصلی برای انتخاب ویژگی وجود دارد: الگوریتم‌های جست‌وجوی کامل، جست‌وجوی تصادفی و جست‌وجوی ابتکاری. در جست‌وجوی کامل تمام راه‌حل‌های ممکن امتحان می‌شوند. در این روش، یافتن جواب بهینه تضمین می‌شود، اما پیچیدگی زمانی مسئله‌ی انتخاب ویژگی، نمایی است (برابر 2^N که N تعداد ویژگی ها در مجموعه داده است). بنابراین استفاده از جست‌وجوی کامل در مسئله انتخاب ویژگی غیرعملی است. جست‌وجوی تصادفی نیز در بدترین حالت، معادل جست‌وجوی کامل خواهد بود. در جست‌وجوی ابتکاری از اطلاعات ابتکاری برای هدایت فرآیند جست‌وجو استفاده می‌شود. این اطلاعات ابتکاری بسیار وابسته به مسئله است و اغلب تعمیم‌ناپذیر. الگوریتم‌های فراابتکاری[3]، فنون جست‌وجوی همه‌منظوره و معمولاً مستقل از مسئله هستند. الگوریتم‌های ژنتیک، ازدحام ذرات و بهینه‌سازی وال مثال‌هایی از الگوریتم‌های فراابتکاری هستند [۵]. الگوریتم‌های فراابتکاری به دلیل ماهیت تصادفی‌شان، نمی‌توانند یافتن مجموعه‌ی ویژگی بهینه را تضمین کنند و مطابق قضیه‌ی Lunch Free No در زمینه بهینه‌سازی، الگوریتمی وجود ندارد که بتواند تمام مسائل بهینه‌سازی را به‌صورت مؤثر حل کند [۳].

در این تحقیق برای بهبود نقاط ضعف هر کدام از الگوریتم‌های فراابتکاری، از ترکیب سه الگوریتم ژنتیک، کلونی مورچگان و بهینه‌سازی وال برای انتخاب ویژگی مبتنی بر پوشش استفاده می‌شود. لازم به ذکر است که در الگوریتم‌های انتخاب ویژگی مبتنی بر پوشش، برای ارزیابی هر زیرمجموعه‌ویژگی از یک طبقه‌بند استفاده می‌گردد و به همین دلیل معمولاً عملکرد بهتری از الگوریتم‌های مبتنی بر فیلتر دارد. انتخاب این سه الگوریتم به دلیل خواص هر کدام از آن‌ها است. فرآیند اکتشاف[4] در الگوریتم ژنتیک بهتر انجام می‌شود. الگوریتم بهینه‌سازی کلونی مورچگان بیشتر روی فرآیند بهره‌برداری[5] تمرکز دارد [۳]. الگوریتم بهینه‌سازی وال تعادلی میان فرآیندهای اکتشاف و بهره‌برداری ایجاد می‌کند [۶]. اجرای الگوریتم‌ها به‌صورت چرخشی و با ترتیب الگوریتم ژنتیک، الگوریتم بهینه‌سازی وال و الگوریتم کلونی مورچگان انجام می‌شود و تعویض بین الگوریتم‌ها بر اساس تعداد راه‌حل‌های بهبود یافته است. پر واضح است که به‌کارگیری روش‌های فوق‌الذکر در راستای پیش‌بینی و کاهش خطاهای نرم‌افزار، یک رویکرد اساسی در طراحی و توسعه‌ی سیستم‌های نرم‌افزاری کیفی محسوب می‌شود. این روش‌ها به نوعی یک رویکرد پدافند غیرعامل در افزایش ارزندگی و ساخت نرم‌افزارهای اقتصادی نیز تلقی می‌گردد.

۲- ادبیات پژوهش

در زمینه پیش‌بینی خطای نرم‌افزار مفاهیمی وجود دارد که برای درک بهتر از تحقیق حاضر در این بخش به بررسی این مفاهیم می‌پردازیم.

۲-۱- طبقه‌بندی[6]

پرکاربردترین فنون مورد استفاده در زمینه‌ی پیش‌بینی خطا، فنون یادگیری ماشین و به‌طور خاص‌تر فنون طبقه‌بندی هستند [۲]. طبقه‌بندی فرآیند تبدیل رکوردهای داده به مجموعه‌ای از کلاس‌ها است. به‌عبارت‌دیگر طبقه‌بندی شامل پیش‌بینی خروجی بر اساس ورودی است. به هر کدام از این رکوردهای داده، یک نمونه[7] گفته می‌شود. رکوردهای داده از تعدادی ویژگی به‌عنوان ورودی و یک برچسب[8] به‌عنوان خروجی تشکیل می‌شوند. هر ویژگی یک خصیصه‌ی قابل اندازه‌گیری از یک پدیده است. برای پیش‌بینی نتایج ابتدا باید داده‌های موجود واکشی شوند. بر اساس این داده‌ها، رکوردها طبقه‌بندی می‌شوند. مجموعه‌داده به دو مجموعه‌ی آموزش[9] و آزمون[10] تقسیم می‌شود. مجموعه‌ی آموزشی شامل داده‌هایی است که قبلاً طبقه‌بندی شده‌اند و به‌عنوان مرجع برای طبقه‌بندی استفاده می‌شوند. مجموعه‌ی آزمون شامل داده‌هایی هستند که هدف یک طبقه‌بند، طبقه‌بندی آن‌ها است. الگوریتم طبقه‌بندی با تحلیل داده‌های موجود در مجموعه داده‌ی آموزش، داده‌های موجود در مجموعه داده‌ی آزمون را طبقه‌بندی می‌کند [۷].

[1] Data Imbalancing
[2] Sampling
[3] Metaheuristic Algorithms
[4] Exploration
[5] Exploitation
[6] Classification
[7] Instance
[8] Tag
[9] Training Dataset
[10] Testing Dataset

۲-۱-۲- جنگل تصادفی

یکی از فنون طبقه‌بندی که به‌طور گسترده در زمینه‌ی پیش‌بینی خطای نرم‌افزار مورد استفاده قرار گرفته و با دیگر فنون مقایسه شده‌است، فن جنگل تصادفی[1] است. جنگل تصادفی یک روش یادگیری جمعی است که از تعداد زیادی درخت تصمیم‌گیری[2] تشکیل می‌شود. در این روش برای مشخص کردن برچسب یک نمونه، از رأی‌گیری میان نتایج هر درخت تصمیم‌گیری استفاده می‌شود. در ادامه به بررسی برخی تحقیقاتی که از جنگل تصادفی برای پیش‌بینی خطای نرم‌افزار استفاده کرده‌اند می‌پردازیم.

گویال و کیشان [۲]، هفده مقاله‌ی پیش‌بینی خطای نرم‌افزار مبتنی بر یادگیری ماشین و برخی فنون مورد استفاده در آن‌ها را مورد بررسی قرار داده‌اند و نتیجه گرفته‌اند که فنون جنگل تصادفی، شبکه‌های عصبی و نایو بیز[3] به اندازه‌ی کافی برای پیش‌بینی خطاهای نرم‌افزار خوب هستند، اما یک فن واحد برای تمامی مجموعه‌داده‌ها مناسب نیست. کریمیان و بابامیر [۴]، هفده طبقه‌بند را روی سیزده مجموعه‌داده‌ی پیش‌بینی خطای نرم‌افزار آزمایش کرده‌اند و نتیجه گرفته‌اند که فنون bagging، جنگل تصادفی و *k بهترین عملکرد را دارند. کائور و کائور [۸]، ۶ فن طبقه‌بندی را روی پنج نرم‌افزار متن باز جاوا آزمایش کرده‌اند. این ۶ فن عبارت است از: جنگل تصادفی، نایو بیز، bagging، درخت تصمیم‌گیری، رگرسیون لاجستیک[4] و IB1. ایشان با مقایسه‌ی این روش‌ها به این نتیجه رسیده‌اند که جنگل تصادفی همیشه بهتر از بقیه و در صدر است و در رتبه‌ی بعدی bagging قرار دارد. سینگ و همکاران [۹]، ۷ فن طبقه‌بندی را روی ۷ پروژه‌ی متن باز آزمایش کرده‌اند. این فنون عبارت است از Adaboost، bagging، درخت تصمیم‌گیری، LogitBoost، جنگل تصادفی و نایو بیز. نتایج آزمایش‌های ایشان نشان می‌دهد که فن جنگل تصادفی بهترین فن در میان فنون ذکر شده است.

نتایج تحقیقات فوق نشان می‌دهد که جنگل تصادفی یکی از بهترین و محبوب‌ترین فنون یادگیری ماشین برای پیش‌بینی خطای نرم‌افزار است. بنابراین در این تحقیق نیز از طبقه‌بند جنگل تصادفی برای پیش‌بینی خطای نرم‌افزار استفاده می‌شود.

جنگل تصادفی دارای پارامترهایی است که مقدار آن‌ها در عملکرد این طبقه‌بند تاثیرگذار است. یکی از مهم‌ترین پارامترهای جنگل تصادفی تعداد متغیرهای کاندیدا در هر انشعاب است که معمولاً با نام mtry شناخته می‌شود. برنارد و همکاران [۱۰] به این نتیجه رسیده‌اند که $mtry = \sqrt{p}$ یک مقدار معقول است، اما گاهی می‌تواند بهبود یابد. لازم به ذکر است که p تعداد کل ویژگی‌ها است. در تحقیق حاضر نیز از همین مقدار برای این پارامتر استفاده می‌گردد. تعداد درختان در جنگل تصادفی یک پارامتر قابل تنظیم نیست. اما باید به اندازه‌ی کافی بزرگ باشد [۱۱]. در این تحقیق برای این پارامتر مقدار ۲۵ در نظر گرفته شده است. به دلیل این که در این تحقیق از کتابخانه‌ی WEKA برای انجام آزمایش‌ها استفاده شده است، برای سایر پارامترهای جنگل تصادفی، مقدار پیش‌فرض آن‌ها در این کتابخانه در نظر گرفته شده است.

۲-۲- انتخاب ویژگی

در سال‌های اخیر، ابعاد بالای داده موجب ایجاد چالش‌های جدی برای روش‌های یادگیری ماشین موجود شده است. یک مدل یادگیری ماشین با وجود تعداد بسیاری از ویژگی‌ها، تمایل به بیش‌برازش پیدا می‌کند که باعث کاهش دقت می‌شود. برای حل این مشکل روی فنون کاهش ابعاد مطالعاتی انجام شده است که یک شاخه‌ی مهم در یادگیری ماشین و داده‌کاوی محسوب می‌شود. انتخاب ویژگی یک فن پرکاربرد در زمینه‌ی کاهش ابعاد است. در انتخاب ویژگی یک زیرمجموعه‌ی کوچک از ویژگی‌های مرتبط از میان تمامی ویژگی‌ها انتخاب می‌شود که معمولاً منجر به بهبود دقت و کاهش هزینه‌ی محاسباتی می‌شود [۱۲].

اغلب منابع، الگوریتم‌های انتخاب ویژگی را از نظر معیار ارزیابی هر زیرمجموعه ویژگی به دو دسته‌ی اصلی الگوریتم‌های مبتنی بر فیلتر[5] و مبتنی بر پوشش[6] تقسیم کرده‌اند [۱۲-۱۴]. برخی از منابع، الگوریتم‌های ترکیبی[7] [۱۳, ۱۴] و الگوریتم‌های تعبیه شده[8] [۱۲, ۱۴] را به عنوان دسته‌های دیگر الگوریتم‌های انتخاب ویژگی در نظر گرفته‌اند.

الگوریتم‌های مبتنی بر فیلتر ارتباط ویژگی‌ها را بر اساس برخی معیارهای متمایزکننده که از مشخصات داده‌ها به دست می‌آیند، ارزیابی می‌کنند. در این نوع از انتخاب ویژگی، از هیچ الگوریتم یادگیری استفاده نمی‌شود. الگوریتم‌های مبتنی بر فیلتر از دو گام تشکیل می‌شوند. در گام اول ویژگی‌ها بر اساس معیارهای خاصی رتبه‌بندی می‌شوند و در گام دوم ویژگی‌هایی که بهترین رتبه‌ها را دارند، انتخاب می‌شوند [۱۵].

[1] Random Forest
[2] Decision Tree
[3] Naïve Bayes
[4] Logistic Regression
[5] Filter-Based Feature Selection
[6] Wrapper-Based Feature Selection
[7] Hybrid Feature Selection
[8] Embedded Feature Selection

الگوریتم‌های مبتنی بر پوشش از یک طبقه‌بند خاص برای ارزیابی کیفیت ویژگی‌های انتخاب شده استفاده می‌کنند و یک راه ساده و قدرتمند برای حل مسئله‌ی انتخاب ویژگی ارائه می‌دهند. در این الگوریتم‌ها از یک طبقه‌بند خاص به‌عنوان یک جعبه سیاه استفاده می‌شود. الگوریتم‌های مبتنی بر پوشش از دو گام تکراری تشکیل می‌شوند. در گام اول زیرمجموعه ویژگی‌ها جستجو می‌شوند و در گام دوم زیرمجموعه ویژگی‌های انتخاب شده با استفاده از عملکرد طبقه‌بند انتخاب شده ارزیابی می‌شوند. این گام‌ها تکرار می‌شوند تا وقتی کیفیت مطلوب به دست آید [۱۲]. به طور کلی الگوریتم‌های مبتنی بر پوشش نتایج بهتری در مقایسه با الگوریتم‌های مبتنی بر فیلتر دارند، اما به دلیل استفاده از الگوریتم‌های یادگیری کندتر هستند [۳].

الگوریتم‌های ترکیبی سعی در استفاده از مزایای هر دو دسته الگوریتم‌های مبتنی بر فیلتر و الگوریتم‌های مبتنی بر پوشش دارند. در این الگوریتم‌ها ابتدا از یک الگوریتم مبتنی بر فیلتر برای کاهش فضای جست‌وجو استفاده شده و در ادامه یک الگوریتم مبتنی بر پوشش برای یافتن بهترین زیرمجموعه ویژگی به‌کار گرفته می‌شود [۱۳, ۱۴]. در الگوریتم‌های تعبیه شده، انتخاب ویژگی در فرآیند ساخت مدل انجام می‌شود [۱۴].

۲-۳- الگوریتم‌های فراابتکاری

الگوریتم‌های فراابتکاری سطح بالاتری از الگوریتم‌های ابتکاری هستند که برای یافتن راه حل‌های مسائل بهینه‌سازی استفاده می‌شوند [۳]. این الگوریتم‌ها فنون جست‌وجوی همه منظوره‌ای هستند که معمولاً مستقل از مسئله هستند [۵]. الگوریتم‌های فراابتکاری دو فاز اصلی به نام‌های اکتشاف و بهره‌برداری دارند. در فاز اکتشاف راه‌حل‌های مختلف به منظور اکتشاف فضای جست‌وجو برای پیدا کردن بهینه‌ی سراسری یافت می‌شوند. در فاز بهره‌برداری جست‌وجوی محلی با استفاده از اطلاعات بهترین راه‌حل‌های یافت شده‌ی اخیر انجام می‌شود [۶]. الگوریتم‌های فراابتکاری به دو دسته‌ی جمعیت‌محور[1] و تک‌راه‌حلی[2] تقسیم می‌شوند. الگوریتم‌های تک‌راه‌حلی تنها یک راه حل را به کار می‌گیرند و تلاش می‌کنند این راه حل را در مجاورت آن بهبود دهند. در مقابل، الگوریتم‌های جمعیت‌محور، فرآیند جست‌وجو را با نگهداری چندین راه حل در نقاط متفاوتی از فضای جست‌وجو هدایت می‌کنند [۱۶].

[1] Population-based
[2] Single-Solution-based

۲-۳-۱- الگوریتم ژنتیک

الگوریتم ژنتیک یک الگوریتم بهینه‌سازی جمعیت‌محور است که فرآیند انتخاب طبیعی را شبیه‌سازی می‌کند. در این الگوریتم، فرآیند بهینه‌سازی با تولید یک مجموعه راه‌حل به نام جمعیت اولیه آغاز می‌شود. سپس عملگرهای ژنتیک شامل انتخاب[3]، تقاطع[4] و جهش[5] روی راه حل‌های انتخاب شده از جمعیت اعمال می‌شود. این گام مرتباً اجرا می‌شود تا شرط خاتمه را ارضا کند یا تکرارها به یک تعداد معین برسد. در هر تکرار، راه‌حل‌ها با استفاده از یک تابع برازش یا هزینه ارزیابی می‌گردند.

تقاطع و جهش عملگرهای تکاملی اصلی در الگوریتم ژنتیک هستند. تولید راه‌حل‌های جدید با انتخاب دو والد از جمعیت بر اساس یک مکانیزم انتخاب، آغاز می‌شود. سپس عملگر تقاطع روی این دو والد اعمال شده و دو فرزند تولید می‌شود. عملگر جهش تغییرات محلی تصادفی در هر دو فرزند ایجاد می‌کند. سپس جمعیت با جای‌گذاری دو فرزند با برخی از راه‌حل‌های موجود در جمعیت که بر اساس یک روش جای‌گذاری انتخاب می‌شوند، به روز می‌شود. سپس الگوریتم با جمعیت به روز شده وارد تکرار بعدی می‌شود [۳].

۲-۳-۲- الگوریتم بهینه‌سازی وال

الگوریتم بهینه‌سازی وال از سوی میرجلیلی و لوئیس [۱۷] ارائه شده است و یک الگوریتم بهینه‌سازی جمعیت‌محور است که رفتار شکاری وال‌های گوژپشت[6] را شبیه‌سازی می‌کند. وال‌های گوژپشت دور طعمه درون دایره‌های کوچک‌شونده شنا می‌کنند و حباب‌های نشانگری در یک مسیر به شکل ۹ ایجاد می‌کنند. در این الگوریتم فاز بهره‌برداری با محاصره‌ی طعمه و روش حمله‌ی شبکه‌ی حبابی حلزونی[7] انجام می‌شود. برای فاز اکتشاف از فرآیند جست‌وجوی تصادفی برای طعمه استفاده می‌گردد.

تعدادی از محققان به بررسی الگوریتم بهینه‌سازی وال در مسائل مختلف از جمله مسئله‌ی انتخاب ویژگی پرداخته‌اند و نتیجه گرفته‌اند که این الگوریتم نسبت به برخی الگوریتم‌های دیگر از جمله ازدحام ذرات عملکرد بهتری دارد [۶, ۱۸, ۱۹].

۲-۳-۳- الگوریتم بهینه‌سازی کلونی مورچگان

الگوریتم بهینه‌سازی کلونی مورچگان یک الگوریتم بهینه‌سازی جمعیت‌محور است که از سوی دوریگو و همکاران [۲۰] ارائه شده است. این الگوریتم رفتار مورچه‌های واقعی را در یافتن کوتاه‌ترین

[3] Selection
[4] Crossover
[5] Mutation
[6] Humpback Whales
[7] Spiral Bubble Net

مسیر شبیه‌سازی می‌کند. حرکات مورچه‌ها بر اساس ماده‌ای به نام فرومون[1] است که بعضی مورچه‌های دیگر روی بعضی از مسیرها می‌ریزند. در این الگوریتم، مورچه‌ها جواب‌های احتمالی را از طریق پیمایش گرافی به نام گراف ساختاری[2] تولید می‌کنند. مورچه‌ها برای تولید جواب کاندید در راستای یال‌های گراف از یک رأس به رأس دیگر حرکت می‌کنند و از این طریق به طور تدریجی یک جواب کاندید جزئی[3] تولید می‌کنند. علاوه بر این، مورچه‌ها در هنگام حرکت در محیط، روی یال‌ها یا رأس‌هایی که در گراف ساختاری پیمایش می‌کنند، مقدار مشخصی فرومون می‌ریزند.

۲-۴- ترکیب الگوریتم‌های فراابتکاری

الگوریتم‌های فراابتکاری روش‌های جست‌وجوی احتمالی هستند که عملکرد بهتری از دیگر روش‌ها در حل مسئله‌ی انتخاب ویژگی دارند [۲۱]. اما به دلیل ماهیت آن‌ها، این الگوریتم‌ها نمی‌توانند یافتن جواب بهینه را تضمین کنند و مطابق قضیه‌ی Lunch Free No در زمینه‌ی بهینه‌سازی، هیچ الگوریتمی وجود ندارد که بتواند تمام مسائل بهینه‌سازی را به طور مؤثر حل کند [۳].محققان برای بهبود نقاط ضعف هر یک از الگوریتم‌های فراابتکاری، از ترکیب آن‌ها استفاده می‌نمایند. به عنوان مثال ترابیه و همکاران [۳]، برای حل مسئله‌ی انتخاب ویژگی از ترکیب سه الگوریتم فراابتکاری ژنتیک، ازدحام ذرات و کلونی مورچگان استفاده کرده‌اند. انتخاب این سه الگوریتم به دلیل خواص هرکدام از آن‌ها است. فرآیند اکتشاف در الگوریتم ژنتیک بهتر از الگوریتم‌های ازدحام ذرات و کلونی مورچگان انجام می‌شود. الگوریتم ازدحام ذرات تعادلی میان اکتشاف و بهره‌برداری برقرار می‌کند. الگوریتم کلونی مورچگان بیشتر روی فرآیند بهره‌برداری تمرکز دارد.

برای تعویض الگوریتم‌ها در هر تکرار از الگوریتم‌های فراابتکاری ترکیبی، می‌توان از دو روش تعویض تصادفی و تعویض تطبیقی استفاده کرد. ترابیه و همکاران [۳] از روش تعویض تصادفی استفاده کرده‌اند. در هر تکرار از الگوریتم پیشنهادی آن‌ها یکی از الگوریتم‌های ژنتیک، ازدحام ذرات و کلونی مورچگان به‌صورت تصادفی انتخاب و اجرا می‌شود. مفرجا و همکاران [۵]، از ترکیب دو الگوریتم بهینه‌سازی وال و گرگ خاکستری برای انتخاب ویژگی استفاده کرده‌اند. آن‌ها برای تعویض الگوریتم‌ها علاوه بر روش تعویض تصادفی از روش تعویض تطبیقی نیز استفاده کرده‌اند. در روش تعویض تطبیقی، بهینه‌سازی با الگوریتم گرگ خاکستری آغاز می‌شود. در هر تکرار تعداد کل راه حل‌های بهبود یافته محاسبه می‌شود. اگر این تعداد کمتر از یک حد آستانه باشد، در تکرار بعدی الگوریتم بهینه‌سازی وال اجرا می‌شود و به همین ترتیب الگوریتم‌ها ممکن است در تکرارهای مختلف تعویض شوند. ایشان نتیجه گرفته‌اند روش تعویض تطبیقی نتایج بهتری در مقایسه با روش تعویض تصادفی دارد. در تحقیق حاضر نیز از ترکیب الگوریتم‌های ژنتیک، بهینه‌سازی وال و کلونی مورچگان با تعویض تطبیقی برای انتخاب ویژگی مبتنی بر پوشش استفاده می‌شود.

۲-۵- پارامترهای الگوریتم‌های فراابتکاری

در این تحقیق از ۴۰ عامل جست‌وجو[4] در الگوریتم ارائه شده استفاده می‌شود. حداکثر تعداد تکرارها در این الگوریتم برابر با ۳۰۰۰ در نظر گرفته شده است. هر کدام از سه الگوریتم بهینه‌سازی استفاده شده در این تحقیق دارای پارامترهایی هستند که مقدار آن‌ها در عملکرد این الگوریتم‌ها موثر است. مقادیر مورد استفاده‌ی پارامترها در این تحقیق در جدول (۱) قابل مشاهده است.

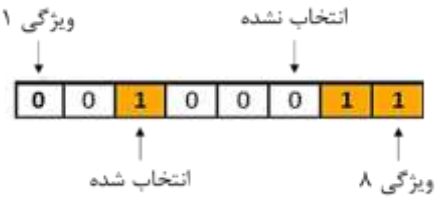
جدول (۱): پارامترهای الگوریتم‌های فراابتکاری

مقدار	پارامتر	الگوریتم
۰/۵	نرخ تقاطع	ژنتیک
۰/۱	نرخ جهش	ژنتیک
چرخ رولت	نوع انتخاب	ژنتیک
۲ تایی	نوع تقاطع	ژنتیک
۱	B	وال
سیگموئید	تابع تبدیل	وال
۱	فرومون اولیه	کلونی مورچگان
۰/۶	نرخ تبخیر	کلونی مورچگان

۲-۶- نمایش راه‌حل و تابع برازش

در طراحی الگوریتم‌های بهینه‌سازی دو موضوع نمایش راه حل و تابع برازش باید مورد توجه قرار گیرد. در این تحقیق راه حل به‌صورت یک بردار دودویی به طول تعداد ویژگی‌ها نمایش داده می‌شود. مقدار صفر به معنی این است که ویژگی انتخاب نشده است و برعکس مقدار ۱ نشان دهنده‌ی انتخاب شدن ویژگی است. نمایش راه حل در شکل ۱ نشان داده شده است.

[1] Pheromone
[2] Construction Graph
[3] Partial Candidate Solution
[4] Search Agent



شکل (۱): نمایش راه حل

در این تحقیق از معیار AUC به‌عنوان تابع برازش استفاده شده است.

۲-۷- نمونه‌برداری

یک مجموعه داده نامتعادل محسوب می‌شود اگر تعداد نمونه‌های یک کلاس در آن بسیار بیشتر از کلاس یا کلاس‌های دیگر باشد. کلاسی که تعداد نمونه‌های آن بیشتر است، کلاس اکثریت[1] و کلاسی که نمونه‌های آن کمتر است کلاس اقلیت[2] نامیده می‌شود. طبقه‌بندی داده‌های نامتعادل چالش‌های زیادی را برای مدل‌های طبقه‌بندی سنتی به همراه دارد [۲۲]. طبقه‌بندهای سنتی برای مجموعه‌های آموزشی متعادل مناسب هستند و در مواجهه با داده‌های نامتعادل نتایج مطلوبی تولید نمی‌کنند، به عنوان مثال نمونه‌های کلاس اکثریت را به خوبی پیش‌بینی می‌کنند، اما دقت خوبی در پیش‌بینی نمونه‌های کلاس اقلیت ندارند. تعداد کم نمونه‌ها و تعداد زیاد ویژگی‌ها اغلب موجب می‌شوند مدل‌های یادگیری در تشخیص الگوهای نادر شکست بخورند [۲۳, ۲۴].

یکی از راه‌های حل مشکل داده‌های نامتعادل، نمونه‌برداری است. روش‌های نمونه‌برداری به دو دسته بیش‌نمونه‌برداری[3] و زیرنمونه‌برداری[4] تقسیم می‌شوند [۲۵]. بیش‌نمونه‌برداری به معنای افزایش تعداد نمونه‌های کلاس اقلیت با تولید نمونه‌های جدید یا تکرار برخی از نمونه‌ها است. زیرنمونه‌برداری به معنای کاهش تعداد نمونه‌های کلاس اکثریت است [۲۶].

۲-۷-۱- نمونه‌برداری به روش SMOTE

نمونه‌برداری SMOTE یک روش بیش‌نمونه‌برداری است که از سوی چاولا و همکاران [۲۵] ارائه شده است. در این نمونه‌برداری، کلاس اقلیت با ایجاد نمونه‌های سنتزی[5] بیش‌نمونه‌برداری می‌شود. به این صورت که یک نمونه از کلاس اقلیت گرفته می‌شود و نمونه‌های سنتزی روی پاره‌خط‌های متصل‌کننده‌ی k همسایه‌ی نزدیک ایجاد می‌شود. با توجه به مقدار مورد نیاز بیش‌نمونه‌برداری، همسایه‌ها از میان k همسایه‌ی نزدیک به‌صورت تصادفی انتخاب می‌شوند. برای مثال اگر مقدار مورد نیاز بیش‌نمونه‌برداری ۲۰۰ درصد باشد، فقط ۲ همسایه از k همسایه‌ی نزدیک انتخاب می‌شود و یک نمونه در جهت هر کدام تولید می‌شود. در این تحقیق برای پارامتر درصد نمونه‌برداری SMOTE عدد ۵۰ در نظر گرفته شده است، اما اگر با درصد نمونه‌برداری ۵۰، تعداد نمونه‌های کلاس اقلیت بعد از نمونه‌برداری از تعداد نمونه‌های کلاس اکثریت بیشتر شود، درصد نمونه‌برداری به نحوی انتخاب می‌شود که پس از نمونه‌برداری، تعداد نمونه‌های کلاس اقلیت با تعداد نمونه‌های کلاس اکثریت برابر شود.

۳- کارهای مرتبط

در سال‌های اخیر، محققان تحقیقات زیادی در زمینه‌ی انتخاب ویژگی و پیش‌بینی خطای نرم‌افزار داشته‌اند. بعضی از محققان برای حل مشکل ابعاد بالای داده از الگوریتم‌های انتخاب ویژگی مبتنی بر فیلتر و بعضی دیگر از الگوریتم‌های مبتنی بر پوشش استفاده کرده‌اند. در ادامه برخی از تحقیقاتی که در زمینه‌ی پیش‌بینی خطای نرم‌افزار از الگوریتم‌های انتخاب ویژگی مبتنی بر پوشش استفاده کرده‌اند، بررسی می‌شوند.

آنبو و مالا [۳۱]، با استفاده از ۳ روش طبقه‌بندی نایو بیز، ماشین بردار پشتیبان و K-NN، مدل‌هایی برای پیش‌بینی خطاهای نرم‌افزار ساخته و از الگوریتم کرم شب‌تاب به‌عنوان روش جست‌وجو در انتخاب ویژگی مبتنی بر پوشش استفاده نموده‌اند. آن‌ها نتیجه گرفته‌اند که الگوریتم انتخاب ویژگی پیشنهادی دقت مدل‌های پیش‌بینی خطا را افزایش می‌دهد.

مانجلا و فلورنس [۳۲]، مدلی برای پیش‌بینی خطاهای نرم‌افزار با استفاده از فن شبکه‌ی عصبی عمیق ساخته و برای انتخاب ویژگی مبتنی بر پوشش از الگوریتم ژنتیک به‌عنوان روش جست‌وجو استفاده کرده‌اند. آن‌ها نتیجه گرفته‌اند که روش پیشنهادی برای پیش‌بینی خطاها، روش مؤثری است.

تومار و همکاران [۳۳]، برای پیش‌بینی خطای نرم‌افزار از سه طبقه‌بند K-NN، درخت تصمیم‌گیری و تحلیل تشخیص خطی[6] استفاده کرده‌اند. آن‌ها برای انتخاب ویژگی مبتنی بر پوشش از الگوریتم بهبود یافته‌ی بهینه‌سازی شعله‌ی پروانه به‌عنوان روش جست‌وجو و برای حل مشکل داده‌های نامتعادل از نمونه‌برداری Adaptive sampling synthetic استفاده نموده‌اند. ایشان نتیجه گرفته‌اند که بهینه‌سازی شعله پروانه بهبودیافته عملکرد طبقه‌بندها را بهبود می‌دهد.

[1] Majority Class
[2] Minority Class
[3] Oversampling
[4] Undersampling
[5] Synthetic Samples
[6] Linear Discriminant Analysis

طاهر و آرمان [۳۴]، مدل‌هایی برای پیش‌بینی خطای نرم‌افزار با استفاده از فنون K-NN، درخت تصمیم‌گیری و تحلیل تشخیص خطی ساخته‌اند. آن‌ها برای انتخاب ویژگی مبتنی بر پوشش از الگوریتم بهینه‌سازی شاهین هریس بهبودیافته به عنوان روش جست‌وجو و برای متعادل کردن داده‌ها از روش نمونه‌برداری sampling synthetic Adaptive استفاده نموده‌اند. ایشان نتیجه گرفته‌اند که الگوریتم بهینه‌سازی شاهین هریس بهبود یافته نسبت به نسخه‌ی اصلی آن در مسئله‌ی انتخاب ویژگی بهتر عمل می‌کند.

ترابیه و همکاران [۳]، یک مدل پیش‌بینی خطا با استفاده از شبکه‌ی عصبی بازگشتی ارائه کرده‌اند. آن‌ها از ترکیب سه الگوریتم فراابتکاری ژنتیک، کلونی مورچگان و ازدحام ذرات با تعویض تصادفی بین آن‌ها در هر تکرار، به عنوان روش جست‌وجو در انتخاب ویژگی مبتنی بر پوشش استفاده کرده‌اند. آن‌ها نتیجه گرفته‌اند که انتخاب ویژگی در ساخت یک طبقه‌بند با کیفیت اهمیت زیادی دارد و روش پیشنهادی از روش‌های موجود بهتر عمل می‌کند.

۴- روش پیشنهادی

فرآیند کلی روش پیشنهادی در شکل (۲) نشان داده شده است. همان طور که مشاهده می‌کنید ابتدا برای حذف ویژگی‌های غیرمفید، عمل انتخاب ویژگی روی مجموعه داده انجام و در نهایت مدل پیش‌بینی خطا روی مجموعه داده‌ی بهینه‌شده ساخته می‌شود.

شکل (۲). فرآیند کلی روش پیشنهادی

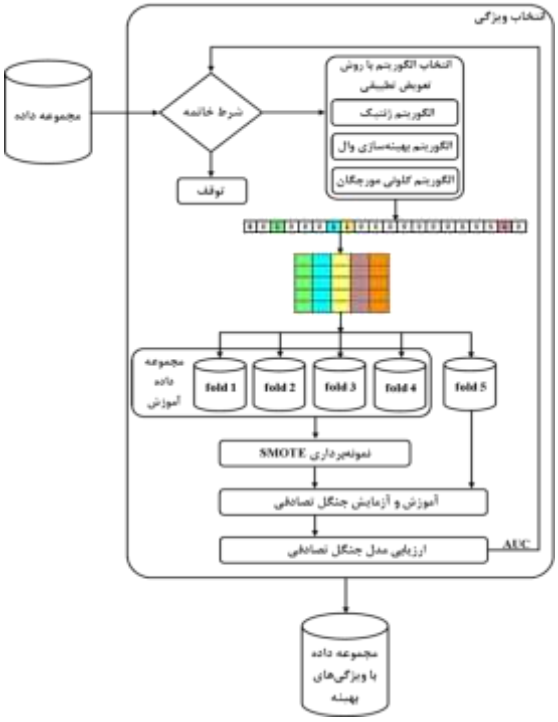
الگوریتم انتخاب ویژگی پیشنهادی در این مقاله یک الگوریتم مبتنی بر پوشش است که از ترکیب سه الگوریتم بهینه‌سازی ژنتیک، وال و کلونی مورچگان به‌عنوان روش جست‌وجوی راه‌حل‌ها استفاده می‌کند. از این پس برای سادگی کار، روش پیشنهادی را GWAS-FSM (Ant Whale Genetic Method Selection Feature Sampling with) می‌نامیم.

۴-۱- روش GWAS-FSM

در شکل (۳) روند نمای روش GWAS-FSM نشان داده شده است. در این روش ابتدا ۴۰ عامل جست‌وجو به صورت تصادفی تولید می‌شوند. انتخاب ویژگی با الگوریتم ژنتیک روی جمعیت اولیه آغاز می‌شود. هر راه‌حل یک زیرمجموعه از مجموعه ویژگی‌ها است. به عنوان مثال اگر در یک مرحله از جست‌وجوی راه‌حل‌ها، مجموعه‌ی A={2, 5, 8} به عنوان یک راه‌حل تولید شود، به معنای آن است که بر اساس این راه‌حل فقط ویژگی‌های شماره‌ی ۲، ۵ و ۸ ویژگی‌های بهینه هستند و سایر ویژگی‌ها باید از مجموعه‌ی داده حذف شوند. برای محاسبه‌ی مقدار برازش هر راه‌حل ابتدا ویژگی‌های ناموجود در آن از مجموعه داده حذف می‌شود، سپس یک مدل پیش‌بینی روی مجموعه داده با استفاده از طبقه‌بند جنگل تصادفی ساخته شده و با فن cross-validation 5-fold ارزیابی گشته و تابع برازش محاسبه می‌گردد. منظور از cross-validation 5-fold این است که ابتدا مجموعه داده به ۵ بخش مساوی تقسیم و سپس ۵ مدل پیش‌بینی ساخته می‌شود. در هر مدل یکی از این ۵ بخش به عنوان مجموعه داده‌ی آزمون و ۴ بخش دیگر به‌عنوان مجموعه داده‌ی آموزش استفاده می‌شود. در نهایت مقدار AUC برای هر یک از این مدل‌ها محاسبه شده و میانگین آن‌ها به عنوان مقدار برازش راه‌حل در نظر گرفته می‌شود. لازم به ذکر است که در هر تکرار از cross-validation نمونه‌برداری به روش SMOTE روی مجموعه داده‌ی آموزش انجام می‌شود. برای تعویض بین الگوریتم‌های فراابتکاری از روش تعویض تطبیقی استفاده می‌شود. به این صورت که در هر تکرار از الگوریتم انتخاب ویژگی اگر تعداد کل راه‌حل‌های بهبود یافته کمتر از یک حد آستانه باشد، در تکرار بعدی الگوریتم بعدی اجرا می‌شود. اجرای الگوریتم‌ها به صورت چرخشی و با ترتیب الگوریتم ژنتیک، الگوریتم بهینه‌سازی وال و الگوریتم کلونی مورچگان انجام می‌شود. حد آستانه‌ی یاد شده یک عدد ثابت نیست. این حد آستانه، در ابتدا برابر با ۴ است و پس از ۵ تکرار به تدریج به نصف تعداد راه حل‌های تغییر یافته می‌رسد. شرط خاتمه این الگوریتم این است که برازش یکی از راه‌حل‌ها به یک مقدار بهینه یا تعداد تکرارها به حداکثر برسد. خروجی این الگوریتم بهترین زیرمجموعه ویژگی پیدا شده از نظر مقدار برازش است که باید ویژگی‌های ناموجود در این زیرمجموعه ویژگی از مجموعه داده حذف شود تا مجموعه داده با ویژگی‌های بهینه به‌دست آید.

۵- ارزیابی روش پیشنهادی

در این بخش نتایج روش GWAS-FSM مورد بحث و بررسی قرار می‌گیرد. برای ارزیابی این روش، ۱۹ مجموعه داده انتخاب شده و مورد آزمایش قرار گرفته است. برای اطمینان بیشتر نسبت به نتایج، روش GWAS-FSM روی هر کدام از مجموعه داده‌های مورد استفاده ۱۰ بار مورد آزمایش قرار گرفته است.



شکل (۳): روندنمای GWAS-FSM

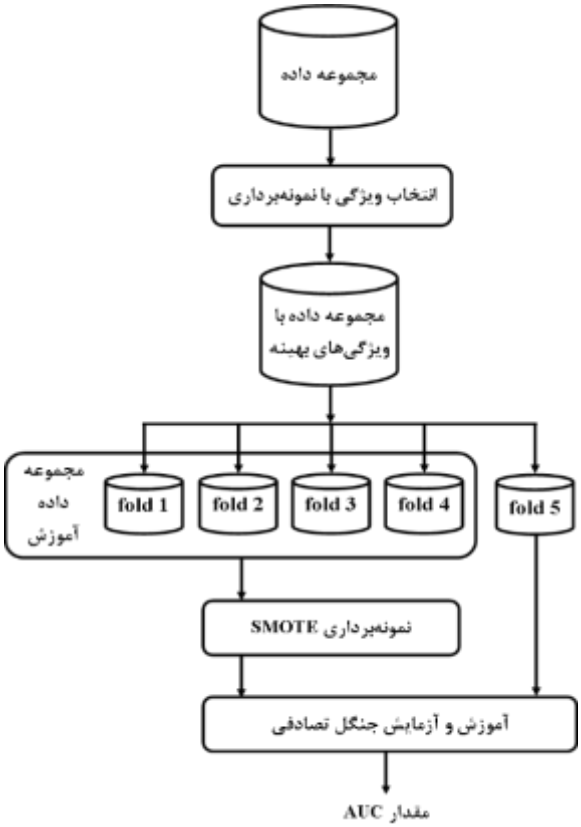
برای ارزیابی روش GWAS-FSM، پس از پایان الگوریتم انتخاب ویژگی، ویژگی‌های انتخاب نشده از مجموعه داده حذف می‌شوند و سپس مدل نهایی پیش‌بینی خطا با استفاده از طبقه‌بند جنگل تصادفی ساخته شده و با فن cross-validation 5-fold ارزیابی می‌گردد. در هر تکرار از cross-validation نمونه‌برداری به روش SMOTE روی مجموعه داده آموزش انجام می‌شود. نحوه‌ی ارزیابی روش GWAS-FSM در شکل (۴) نشان داده شده است.

۵-۱- مجموعه داده‌های مورد استفاده

چندین مجموعه داده عمومی مانند:

مجموعه‌داده‌های پرامیس[1]، ناسا[2] و AEEEM در زمینه‌ی پیش‌بینی خطای نرم‌افزار وجود دارند. در این تحقیق ۱۹ پروژه‌ی نرم‌افزاری واقعی از مخزن عمومی مهندسی نرم‌افزار پرامیس برای انجام آزمایش‌ها انتخاب شده‌اند. این مجموعه‌داده‌ها از سوی چندین محقق در مهندسی نرم‌افزار بسیار توصیه شده‌اند. پروژه‌های انتخاب شده، اندازه‌های مختلف (۱۰۹ تا ۹۰۹ نمونه) و درصد متفاوتی از نمونه‌های خطادار (از ۲/۲ تا ۹۸/۸ درصد) دارند. هر کدام از این مجموعه داده‌ها از ۲۰ ویژگی شیءگرا به‌عنوان ورودی و یک مقدار خطا به‌عنوان خروجی تشکیل شده‌اند. خلاصه مختصری از مجموعه داده‌های انتخابی را در جدول (۲) مشاهده می‌کنید.

[1] PROMISE Datasets
[2] NASA Datasets



شکل (۴): نحوه ارزیابی روش GWAS-FSM

جدول (۲): مجموعه داده‌های مورد استفاده

مجموعه داده	تعداد نمونه‌ها	تعداد نمونه‌های خطادار	نرخ نمونه‌های خطادار
Ant-1.7	۷۴۵	۱۶۶	۰/۲۲۳
Camel-1.0	۳۳۹	۱۳	۰/۰۳۸
Camel-1.2	۶۰۸	۲۱۶	۰/۳۵۵
Camel-1.4	۸۷۲	۱۴۵	۰/۱۶۶
Camel-1.6	۹۶۵	۱۸۸	۰/۱۹۵
Jedit-3.2	۲۷۲	۹۰	۰/۳۳۱
Jedit-4.0	۳۰۶	۷۵	۰/۲۴۵
Jedit-4.2	۳۶۷	۴۸	۰/۱۳۱
Jedit-4.3	۴۹۲	۱۱	۰/۰۲۲
Log4j-1.0	۱۳۵	۳۴	۰/۲۵۲
Log4j-1.1	۱۰۹	۳۷	۰/۳۳۹
Log4j-1.2	۲۰۵	۱۸۹	۰/۹۲۲
Lucene-2.0	۱۹۵	۹۱	۰/۴۶۷
Lucene-2.2	۲۴۷	۱۴۴	۰/۵۸۳
Lucene-2.4	۳۴۰	۲۰۳	۰/۵۹۷
Xalan-2.4	۷۲۳	۱۱۰	۰/۱۵۲
Xalan-2.5	۸۰۳	۳۸۷	۰/۴۸۲
Xalan-2.6	۸۸۵	۴۱۱	۰/۴۶۴
Xalan-2.7	۹۰۹	۸۹۸	۰/۹۸۸

۵-۲- ابزار مورد استفاده

برای پیاده‌سازی روش پیشنهادی از زبان برنامه‌نویسی جاوا و کتابخانه‌ی یادگیری ماشین WEKA استفاده شده است. WEKA ابزار متن بازی است که به دو صورت گرافیکی و کتابخانه‌ای در دسترس عموم قرار دارد [۳۵].

۵-۳- نتایج روش GWAS-FSM

نتایج روش GWAS-FSM را در جدول ۳ مشاهده می‌کنید. در این جدول مقادیر میانگین AUC، کمترین AUC، بیشترین AUC و AUC میانه گزارش شده است.

جدول (۳): نتایج روش GWAS-FSM (بر اساس AUC)

مجموعه داده	میانگین	بیشترین	کمترین	میانه
Ant-1.7	۰/۸۵۳۶	۰/۸۶۰۷	۰/۸۴۹۸	۰/۸۵۳۳
Camel-1.0	۰/۸۸۴۹	۰/۹۲۲۹	۰/۸۶۰۳	۰/۸۸۲۲
Camel-1.2	۰/۷۴۳۵	۰/۷۶۶۱	۰/۷۲۷۳	۰/۷۴۴۸
Camel-1.4	۰/۷۶۸۹	۰/۷۷۶۶	۰/۷۶۲۹	۰/۷۶۸۰
Camel-1.6	۰/۷۶۶۴	۰/۷۷۷۱	۰/۷۵۱۸	۰/۷۶۶۷
Jedit-3.2	۰/۸۷۷۴	۰/۸۹۱۳	۰/۸۶۹۵	۰/۸۷۵۷
Jedit-4.0	۰/۸۶۵۴	۰/۸۸۳۳	۰/۸۵۲۷	۰/۸۶۵۶
Jedit-4.2	۰/۸۸۵۱	۰/۸۹۲۹	۰/۸۷۰۸	۰/۸۸۷۵
Jedit-4.3	۰/۹۲۳۲	۰/۹۶۰۵	۰/۸۹۰۴	۰/۹۲۱۲
Log4j-1.0	۰/۸۸۷۱	۰/۹۰۷۳	۰/۸۶۰۷	۰/۸۹۲۰
Log4j-1.1	۰/۹۰۷۳	۰/۹۱۹۱	۰/۸۷۸۷	۰/۹۱۱۰
Log4j-1.2	۰/۹۳۸۱	۰/۹۵۴۹	۰/۹۱۶۶	۰/۹۳۹۴
Lucene-2.0	۰/۸۰۷۴	۰/۸۲۵۸	۰/۷۸۱۲	۰/۸۰۹۱
Lucene-2.2	۰/۷۴۹۶	۰/۷۸۱۱	۰/۷۲۸۵	۰/۷۴۶۵
Lucene-2.4	۰/۸۳۲۶	۰/۸۴۷۷	۰/۸۰۹۸	۰/۸۳۳۱
Xalan-2.4	۰/۸۴۴۸	۰/۸۵۴۵	۰/۸۳۷۸	۰/۸۴۵۱
Xalan-2.5	۰/۸۰۱۲	۰/۸۱۳۹	۰/۷۹۲۱	۰/۸۰۱۲
Xalan-2.6	۰/۸۷۲۴	۰/۸۷۵۷	۰/۸۶۸۵	۰/۸۷۲۶
Xalan-2.7	۰/۹۹۱۰	۰/۹۹۶۲	۰/۹۸۵۰	۰/۹۹۱۴

۵-۳-۱- مقایسه با حالت بدون انتخاب ویژگی

در جدول ۴ نتایج روش GWAS-FSM با حالت بدون انتخاب ویژگی مقایسه شده است. لازم به ذکر است که در حالت بدون انتخاب ویژگی نیز از طبقه‌بند جنگل تصادفی با پارامترهای مشابه و با فن ارزیابی Cross-Validation 5-Fold استفاده شده است.

همان‌طور که در جدول ۴ مشاهده می‌کنید، انتخاب ویژگی در تمام مجموعه‌داده‌ها منجر به بهبود نتایج می‌شود.

جدول (۴): مقایسه با حالت بدون انتخاب ویژگی (بر اساس AUC)

مجموعه داده	GWAS-FSM	بدون انتخاب ویژگی
Ant-1.7	**۰/۸۵۳۶**	۰/۸۰۸۸
Camel-1.0	**۰/۸۸۴۹**	۰/۵۹۵۸
Camel-1.2	**۰/۷۴۳۵**	۰/۶۷۴۳
Camel-1.4	**۰/۷۶۸۹**	۰/۷۲۱۴
Camel-1.6	**۰/۷۶۶۴**	۰/۷۰۵۳
Jedit-3.2	**0.8774**	۰/۸۳۲۲
Jedit-4.0	**۰/۸۶۵۴**	۰/۸۱۱۴
Jedit-4.2	**۰/۸۸۵۱**	۰/۸۱۲۵
Jedit-4.3	**۰/۹۲۳۲**	۰/۸۲۷۱
Log4j-1.0	**۰/۸۸۷۱**	۰/۷۹۵۴
Log4j-1.1	**۰/۹۰۷۳**	۰/۸۲۵۹
Log4j-1.2	**۰/۹۳۸۱**	۰/۷۴۸۱
Lucene-2.0	**۰/۸۰۷۴**	۰/۷۴۳۴
Lucene-2.2	**۰/۷۴۹۶**	۰/۶۳۰۶
Lucene-2.4	**۰/۸۳۲۶**	۰/۸۰۷۲
Xalan-2.4	**۰/۸۴۴۸**	۰/۸۱۲۹
Xalan-2.5	**۰/۸۰۱۲**	۰/۷۷۴۸
Xalan-2.6	**۰/۸۷۲۴**	۰/۸۵۲۹
Xalan-2.7	**۰/۹۹۱۰**	۰/۸۵۲۰
میانگین	**۰/۸۵۲۶**	۰/۷۷۰۰

جدول (۵): مقایسه با روش‌های انتخاب ویژگی موجود

مجموعه داده	GWAS-FSM	ترابیه و همکاران [۳]	تومار و همکاران [۳۳]			طاهر و آرمان [۳۴]
			K-NN	DT	LDA	
Ant-1.7	۰/۸۵۳۶	**۰/۸۸۴۲**	۰/۷۲۴۵	۰/۶۷۷۲	۰/۷۵۰۵	۰/۷۷۲۷
Camel-1.0	۰/۸۸۴۹	**۰/۹۰۵۲**	۰/۷۰۴۶	۰/۶۳۲۷	۰/۷۳۸۱	۰/۸۱۰۷
Camel-1.2	**۰/۷۴۳۵**	۰/۶۴۸۴	۰/۶۰۰۴	۰/۶۰۳۳	۰/۶۲۸۰	۰/۶۴۶۷
Camel-1.4	۰/۷۶۸۹	**۰/۷۸۰۷**	۰/۶۳۷۸	۰/۶۰۳۵	۰/۶۷۹۰	۰/۷۰۲۹
Camel-1.6	**۰/۷۶۶۴**	۰/۶۹۱۴	۰/۶۴۱۶	۰/۵۹۵۳	۰/۶۵۱۲	۰/۶۷۶۲
Jedit-3.2	۰/۸۷۷۴	**۰/۹۱۷۷**	۰/۷۳۹۳	۰/۶۸۹۹	۰/۸۰۸۲	۰/۸۲۷۰
Jedit-4.0	۰/۸۶۵۴	**۰/۸۷۰۰**	۰/۷۰۵۸	۰/۷۰۳۲	۰/۷۲۲۴	۰/۷۶۶۱
Jedit-4.2	۰/۸۸۵۱	**۰/۸۹۱۳**	۰/۷۵۹۸	۰/۶۷۳۱	۰/۷۵۸۹	۰/۸۲۹۰
Jedit-4.3	**۰/۹۲۳۲**	۰/۹۱۳۱	۰/۷۰۸۲	۰/۶۷۸۶	۰/۶۴۲۶	۰/۸۰۸۱
Log4j-1.0	**۰/۸۸۷۱**	۰/۸۷۵۹	۰/۷۴۲۹	۰/۷۱۶۱	۰/۷۷۸۶	۰/۸۲۹۷
Log4j-1.1	۰/۹۰۷۳	**۰/۹۱۴۵**	۰/۸۰۲۶	۰/۷۴۳۷	۰/۷۸۶۶	۰/۸۳۹۵
Log4j-1.2	**۰/۹۳۸۱**	۰/۸۵۲۲	۰/۷۳۵۹	۰/۷۱۱۷	۰/۶۷۴۹	۰/۷۷۶۱
Lucene-2.0	۰/۸۰۷۴	**۰/۸۶۷۷**	-	-	-	-
Lucene-2.2	۰/۷۴۹۶	**۰/۸۲۳۶**	-	-	-	-
Lucene-2.4	**۰/۸۳۲۶**	۰/۸۲۵۹	-	-	-	-
Xalan-2.4	**۰/۸۴۴۸**	۰/۸۲۷۰	۰/۷۲۲۵	۰/۶۳۰۳	۰/۷۱۹۳	۰/۷۵۲۱
Xalan-2.5	**۰/۸۰۱۲**	۰/۷۴۹۳	-	-	-	-
Xalan-2.6	**۰/۸۷۲۴**	۰/۷۱۲۵	-	-	-	-
Xalan-2.7	**۰/۹۹۱۰**	۰/۹۲۹۵	۰/۸۶۸۱	۰/۸۳۳۵	۰/۸۶۵۰	۰/۹۲۶۹
میانگین	**۰/۸۵۲۶**	۰/۸۳۵۸	-	-	-	-

برای بررسی تأثیر انتخاب ویژگی روی عملکرد طبقه‌بند جنگل تصادفی یک مقایسه‌ی آماری با استفاده از آزمون آماری ویلکاکسون[1] بین روش GWAS-FSM و حالت بدون انتخاب ویژگی انجام شده است. مقدار p-value به دست آمده در این آزمون آماری برابر با ۰/۰۰۰۱۳۲ است که بسیار کمتر از ۰/۰۵ است و بیانگر تفاوت قابل ملاحظه‌ی روش GWAS-FSM با حالت بدون انتخاب ویژگی است.

۵-۳-۲- مقایسه با دیگر روش‌های انتخاب ویژگی

در جدول (۵) نتایج روش GWAS-FSM با دیگر روش‌های انتخاب ویژگی مقایسه شده است. طبق جدول (۵) واضح است که روش GWAS-FSM در تمام مجموعه داده‌ها از روش تومار و همکاران و همچنین روش طاهر و آرمان، عملکرد بهتری دارد. در ۱۰ مورد از ۱۹ مجموعه داده‌ی انتخابی روش GWAS-FSM از روش ترابیه و همکاران عملکرد بهتری دارد. همچنین میانگین این روش نسبت به تحقیق ترابیه و همکاران بیشتر است، بنابراین در مجموع می‌توان گفت که این روش در مقایسه با روش ترابیه و همکاران بهتر عمل می‌کنند.

۵-۳-۳- بهترین خروجی میان ۱۰ اجرا

در جدول (۶) ویژگی‌های انتخاب شده‌ی حاصل از روش GWAS-FSM برای بیشترین AUC به دست آمده در هر مجموعه داده نشان داده شده است. همان طور که در جدول (۶) مشاهده می‌کنید، بهترین ویژگی‌ها برای هر مجموعه داده متفاوت است، بنابراین برای به دست آوردن بهترین ویژگی‌ها برای هر مجموعه داده‌ی جدید، باید انتخاب ویژگی را روی آن مجموعه داده انجام داد. نکته دیگری که از جدول (۶) می‌توان مشاهده کرد، این است که زیرمجموعه ویژگی‌های بهینه برای هر مجموعه داده دارای اندازه‌های متفاوتی نیز است.

۷- تهدیدات اعتبار نتایج

دو تهدید نسبت به نتایج به دست آمده در تحقیق حاضر ممکن است وجود داشته باشد. اول این که در روش GWAS-FSM از الگوریتم‌های فراابتکاری برای جست‌وجوی زیرمجموعه ویژگی‌ها استفاده شده است، بنابراین در هر بار اجرای این روش لزوماً یک زیرمجموعه ویژگی خاص به‌عنوان خروجی تولید نمی‌شود و در هر بار اجرا ممکن است خروجی متفاوت باشد. دوم این که تمام مجموعه داده‌هایی که در تحقیق حاضر مورد آزمایش قرار گرفته‌اند، مربوط به پروژه‌های نوشته‌شده به زبان جاوا هستند، اما انتظار می‌رود روش GWAS-FSM برای زبان‌های برنامه‌نویسی دیگر نیز معتبر باشند. در هر صورت برای اثبات قابل قبول بودن نتایج روش ذکر شده روی مجموعه داده‌های دیگر، بهتر است این روش روی آن‌ها آزمایش شود.

1 Wilcoxon Statistical Test

جدول (۶): خروجی روش GWAS-FSM برای بیشترین AUC

مجموعه داده	ویژگی‌های انتخاب شده	AUC
Ant-1.7	5, 7, 11, 13 ,14, 15, 16	۰/۸۶۰۷
Camel-1.0	7, 11, 12, 16, 19, 20	۰/۹۲۲۹
Camel-1.2	1, 2, 5, 7, 10, 13, 14, 15, 16, 17, 19	۰/۷۶۶۱
Camel-1.4	1, 2, 3, 7, 8, 9, 10, 12, 13, 14	۰/۷۷۶۶
Camel-1.6	2, 3, 5, 7, 8, 9, 10, 13, 14, 16, 17, 18, 19	۰/۷۷۷۱
Jedit-3.2	1, 4, 5, 7, 8, 10, 12, 14, 17, 19	۰/۸۹۱۳
Jedit-4.0	1, 2, 4, 5, 8, 11, 12, 14, 18, 20	۰/۸۸۳۳
Jedit-4.2	3, 7, 10, 11, 13, 16, 17, 20	۰/۸۹۲۹
Jedit-4.3	2, 4, 5, 7, 13, 15, 17, 19, 20	۰/۹۶۰۵
Log4j-1.0	1, 3, 7, 9, 14, 17, 18	۰/۹۰۷۳
Log4j-1.1	5, 14, 15	۰/۹۱۹۱
Log4j-1.2	8, 12, 15, 18	۰/۹۵۴۹
Lucene-2.0	1, 2, 3, 4, 5, 16, 20	۰/۸۲۵۸
Lucene-2.2	2, 3, 6, 8, 17, 19	۰/۷۸۱۱
Lucene-2.4	2, 3, 4, 8, 9, 16, 17, 19	۰/۸۴۷۷
Xalan-2.4	2, 3, 4, 7, 8, 9, 10, 11, 13, 17, 18	۰/۸۵۴۵
Xalan-2.5	2, 3, 6 , 7, 8, 9, 11, 13, 14, 15, 17, 18	۰/۸۱۳۹
Xalan-2.6	2, 4, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18	۰/۸۷۵۷
Xalan-2.7	3, 4, 7, 9, 10, 13, 15, 17, 18, 20	۰/۹۹۶۲

۸- نتیجه‌گیری و کارهای آینده

وقوع خطاها در نرم‌افزارها یک اتفاق رایج محسوب می‌شود. اما شناسایی زودهنگام خطاها برای توسعه‌دهندگان نرم‌افزارها اهمیت بالایی دارد؛ زیرا هر اندازه که خطاها در نرم‌افزار دیرتر شناسایی شوند، تلاش و هزینه‌ی بیشتری برای رفع آن‌ها نیاز خواهد بود. پیش‌بینی خطاهای نرم‌افزار به معنای شناسایی ماژول‌های مستعد خطا در مراحل اولیه‌ی توسعه‌ی نرم‌افزار است. دقت پایین مدل‌های پیش‌بینی خطای نرم‌افزار باعث می‌شود که بعضی از ماژول‌های خطادار به‌موقع شناسایی نشوند و در نتیجه تلاش و هزینه‌ی لازم برای برطرف کردن خطاها افزایش می‌یابد. یکی از مشکلاتی که عملکرد مدل‌های پیش‌بینی خطای نرم‌افزار را تحت تأثیر قرار می‌دهد، ابعاد بالای داده است. ابعاد بالای داده به معنی وجود ویژگی‌های افزونه و غیرمفید است که باعث کاهش دقت مدل‌های پیش‌بینی خطای نرم‌افزار می‌شود. برای حل این مشکل محققان از الگوریتم‌های انتخاب ویژگی استفاده می‌کنند. در این تحقیق یک روش انتخاب ویژگی مبتنی بر پوشش جدید با استفاده از ترکیب الگوریتم‌های فراابتکاری ارائه شد. برای ارزیابی روش ارائه شده، ۱۹ مجموعه داده انتخاب شد که این روش روی هر کدام از این مجموعه داده‌ها ۱۰ بار آزمایش شد و نتایج با

حالت بدون انتخاب ویژگی و همچنین دیگر روش‌های انتخاب ویژگی موجود مقایسه شد. نتایج نشان می‌دهند که روش ارائه شده قادر است عملکرد جنگل تصادفی را بهبود دهند. همچنین مقایسه‌ی نتایج روش ارائه شده با دیگر روش‌های موجود، برتری این روش را نشان می‌دهد. نکته‌ی دیگر این که با مشاهده‌ی بهترین خروجی روش پیشنهادی، برای هر مجموعه داده در میان ۱۰ آزمایش انجام شد، به این نتیجه می‌رسیم که ویژگی‌های بهینه برای هر مجموعه داده متفاوت هستند.

هدف روش انتخاب ویژگی پیشنهادی در این مقاله افزایش دقت پیش‌بینی خطای نرم‌افزار است. برای افزایش بیشتر دقت پیش‌بینی خطای نرم‌افزار می‌توان سه پیشنهاد دیگر را مطرح نمود:

۱. الگوریتم انتخاب ویژگی پیشنهادی، یک الگوریتم مبتنی بر پوشش است که می‌توان آن را با الگوریتم‌های مبتنی بر فیلتر ترکیب کرد و یک الگوریتم انتخاب ویژگی ترکیبی ساخت.

۲. در این تحقیق بهینه‌سازی پارامترهای جنگل تصادفی انجام نشده است. بهینه‌سازی پارامترهای جنگل تصادفی به دو صورت قابل انجام است. روش اول این است که این پارامترها در حین انتخاب ویژگی، بهینه‌سازی شوند که در این صورت باید این پارامترها را وارد نمایش راه حل کرد. روش دوم این است که بهینه‌سازی این پارامترها بعد از اتمام انتخاب ویژگی انجام شود.

۳. در تحقیق حاضر برای جست‌وجوی زیرمجموعه ویژگی‌ها از الگوریتم‌های فراابتکاری استفاده شده است. برای انجام این جست‌وجو می‌توان از رویکرد ابرابتکاری[1] استفاده نمود. در این رویکرد، یک زمان‌بند به صورت قطعی یا غیر قطعی یک مجموعه از جست‌وجوهای ابتکاری را زمان‌بندی می‌کند. حالت پیچیده‌تر در این رویکرد استفاده از جست‌وجوی فراابتکاری برای انجام این زمان‌بندی است [۳۶].

۸- مراجع

[1] J. Gaur, A. Goyal, T. Choudhury, and S. Sabitha, "A Walk Through of Software Testing Techniques," in 5th International Conference on System Modeling & Advancement in Research Trends, Moradabad, 2016.

[2] J. Goyal and B. Kishan, "Progress on Machine Learning Techniques for Software Fault Prediction," International Journal of Advanced Trends in Computer Science and Engineering, vol. 8, no. 2, pp. 305-311, 2019.

[3] H. Turabieh, M. Mafarja, and X. Li, "Iterated feature selection algorithms with layered recurrent neural network for software fault prediction," Expert Systems With Applications, vol. 122, pp. 27-42, 2019.

[4] F. Karimian and S. M. Babamir, "Evaluation of Classifiers in Software Fault-Proneness Prediction," Journal of AI and Data Mining, vol. 5, no. 2, pp. 149-167, 2017.

[5] M. Mafarja, A. Qasem, A. A. Heidari, I. Aljarah, H. Faris, and S. Mirjalili, "Efficient Hybrid Nature-Inspired Binary Optimizers for Feature Selection," Cognitive Computation, vol. 12, no. 1, pp. 150-175, 2019.

[6] H. M. Mohammad, S. U. Umar, and T. A. Rashid, "A Systematic and Meta-Analysis Survey of Whale Optimization Algorithm," Computational Intelligence and Neuroscience, vol. 2019, pp. 1-25, 2019.

[7] S. Umadevi and K. S. J. Marseline, "A Survey on Data Mining Classification Algorithms," in International Conference on Signal Processing and Communication, Karunya Nagar, 2017.

[8] A. Kaur and I. Kaur, "An empirical evaluation of classification algorithms for fault prediction in open source projects," Journal of King Saud University - Computer and Information Sciences, vol. 30, no. 1, pp. 2-17, 2018.

[9] P. Singh, R. Malhotra, and S. Bansal, "Analyzing the Effectiveness of Machine Learning Algorithms for Determining Faulty Classes: A Comparative Analysis," in 9th International Conference on Cloud Computing, Data Science & Engineering, Noida, 2019.

[10] S. Bernard, L. Heutte, and S. Adam, "Influence of Hyperparameters on Random Forest Accuracy," in International Workshop on Multiple Classifier Systems, Reykjavik, 2009.

[11] E. Scornet, "Tuning parameters in random forests," ESAIM: Proceedings and surveys, vol. 60, pp. 144-162, 2018.

[12] B. Venkatesh and J. Anuradha, "A Review of Feature Selection and Its Methods," Cybernetics and Information Technologies, vol. 19, no. 1, pp. 3-26, 2019.

[13] N. Mlambo, W. K. Cheruiyot, and M. W. Kimwele, "A Survey and Comparative Study of Filter and Wrapper Feature Selection Techniques," The International Journal Of Engineering And Science, vol. 5, no. 8, pp. 57-67, 2016.

[1] Hyper Heuristic Approach

[14] A. Jović, K. Brkić, and N. Bogunović, "A review of feature selection methods with applications," in 2015 38th international convention on information and communication technology, electronics and microelectronics (MIPRO), 2015: Ieee, pp. 1200-1205.

[15] A. O. Balogun, S. Basri, S. J. Abdulkadir, and A. S. Hashim, "Performance Analysis of Feature Selection Methods in Software Defect Prediction: A Search Method Approach," applied sciences, vol. 9, no. 13, p. 2764, 2019.

[16] T. Dokeroglu, E. Sevinc, T. Kucukyilmaz, and A. Cosar, "A survey on new generation metaheuristic algorithms," Computers & Industrial Engineering, vol. 137, pp. 1-29, 2019.

[17] S. Mirjalili and A. Lewis, "The Whale Optimization Algorithm," Advances in Engineering Software, vol. 95, pp. 51-67, 2016.

[18] M. M. Mafarja and S. Mirjalili, "Whale Optimization Approaches for Wrapper Feature Selection," Applied Soft Computing, vol. 62, pp. 441-453, 2018.

[19] M. Sharawi, H. M. Zawbaa, and E. Emary, "Feature Selection Approach Based on Whale Optimization Algorithm," in Ninth International Conference on Advanced Computational Intelligence, Doha, 2017.

[20] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," IEEE Computational Intelligence Magazine, vol. 1, no. 4, pp. 28-39, 2006.

[21] E. Zorarpacı and S. A. Özel, "A hybrid approach of differential evolution and artificial bee colony for feature selection," Expert Systems with Applications, vol. 62, pp. 91-103, 2016.

[22] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, and G. Bing, "Learning from class-imbalanced data: Review of methods and applications," Expert Systems With Applications, vol. 73, pp. 220-239, 2017.

[23] V. López, A. Fernández, S. García, V. Palade, and F. Herrera, "An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics," Information Sciences, vol. 250, pp. 113-141, 2013.

[24] P. Branco, L. Torgo, and R. P. Ribeiro, "A Survey of Predictive Modeling on Imbalanced Domains," ACM Computing Surveys, vol. 49, no. 2, pp. 1-50, 2016.

[25] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," Journal of Artificial Intelligence Research, vol. 16, pp. 321-357, 2002.

[26] R. Mohammed, J. Rawashdeh, and M. Abdullah, "Machine Learning with Oversampling and Undersampling Techniques: Overview Study and Experimental Results," in 11th International Conference on Information and Communication Systems, Irbid, 2020.

[27] C. Catal and B. Diri, "Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem," Information Sciences, vol. 179, no. 8, pp. 1040-1058, 2009.

[28] E. Borandag, A. Ozcift, D. Kilinc, and F. Yucalar, "Majority Vote Feature Selection Algorithm in Software Fault Prediction," Computer Science and Information Systems, vol. 16, no. 2, pp. 515-539, 2019.

[29] A. K. Jakhar and K. Rajnish, "Software fault prediction with data mining techniques by using feature selection based models," International Journal on Electrical Engineering and Informatics, vol. 10, no. 3, pp. 447-465, 2018.

[30] S. Jacob and G. Raju, "Software Defect Prediction in Large Space Systems through Hybrid Feature Selection and Classification," The International Arab Journal of Information Technology, vol. 14, no. 2, pp. 208-214, 2017.

[31] M. Anbu and G. S. A. Mala, "Feature selection using firefly algorithm in software defect prediction," Cluster Computing, vol. 22, no. 5, pp. 10925-10934, 2019.

[32] C. Manjula and L. Florence, "Deep neural network based hybrid approach for software defect prediction using software metrics," Cluster Computing, vol. 22, no. 4, pp. 9847-9863, 2018.

[33] I. Tumar, Y. Hassouneh, H. Turabieh, and T. Thaher, "Enhanced Binary Moth Flame Optimization as a Feature Selection Algorithm to Predict Software Fault Prediction," IEEE Access, vol. 8, pp. 8041-8055, 2020.

[34] T. Thaher and N. Arman, "Efficient Multi-Swarm Binary Harris Hawks Optimization as a Feature Selection Approach for Software Fault Prediction," in 11th International Conference on Information and Communication Systems, Irbid, 2020.

[35] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA Data Mining Software: An Update," SIGKDD Explorations, vol. 11, no. 1, pp. 10-18, 2009.

[36] E. Özcan, B. Bilgin, and E. E. Korkmaz, "A comprehensive analysis of hyper-heuristics," Intelligent data analysis, vol. 12, no. 1, pp. 3-23, 2008.