Faculty of Engineering and Technology

Master of Software Engineering

Ensemble Feature Selection Metaheuristics Algorithms with Layered-Recurrent Neural Network for Software Fault Prediction

---

Author:
Subhi Maraa'beh (1155401)

Supervisor:
Dr. Majdi Mafarja

A thesis submitted in fulfillment of the requirements for the degree of Master of Science in Software Engineering at Birzeit University, Palestine

January 31, 2019

Faculty of Engineering and Technology

Master of Software Engineering

Master Thesis

Ensemble Feature Selection Metaheuristics Algorithms with Layered-Recurrent Neural Network for Software Fault Prediction

تجميع خوارزميات اختيار الميزات مع الشبكات العصبية الطبقية المتكررة للتنبؤ بأعطال البرامج

Author
Student Name: Subhi Maraa'beh
Student Number: 1155401

Supervisor
Dr. Majdi Mafarja

Committee:
Dr. Majdi Mafarja
Dr. Abdel Salam Sayyad
Dr. Sobhi Ahmed

This thesis was submitted in partial fulfillment of the requirements for the Master's Degree in software engineering from the Faculty of Graduate Studies, at Birzeit University, Palestine

Ensemble Feature Selection Metaheuristics Algorithms with Layered-Recurrent Neural Network for Software Fault Prediction

By:

Subhi Maraa'beh

This thesis was prepared under the supervision of Dr. Majdi Mafarja and has been approved by all members of the thesis examination committee

Approved by thesis committee

Dr. Majdi Mafarja (Chair), Birzeit University 

Dr. Abdel Salam Sayyad (Member), Birzeit University

Dr. Sobhi Ahmed (Member), Birzeit University

Date Approved:____3/5/2021_____

Declaration of Authorship

I, Subhi Maraa'beh (1155401), declare that this thesis titled, "Ensemble Feature Selection Metaheuristics Algorithms with Layered-Recurrent Neural Network for Software Fault Prediction" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a master degree at Birzeit University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

_____

Date:

_____

Acknowledgements

First of all I would thank Allah for getting this work done and my thesis advisor Dr. Majdi Mafarja from the Department of Computer Science / faculty of Engineering and Technology at Birzeit University. Dr. Majdi was always there when I needed him, whether I ran into a problem or needed a question Dr. Majdi was ready with the curing answer and advice.

This work couldn't be achieved without the support of my beloved wife Razan and her constant encouragement throughout my years of study, through the process of researching and writing this thesis, nights and nights she was able to absorb all the nervousness I went through.

I dedicate this success to my children Omar and Reem and whoever is coming down the road, to my parents Abu/Um Yazan, brothers, sister and family, to my friends all of them, and especially Ehab, Marwan and my partner Ali. This accomplishment would not have been possible without them. Thank you.

Author

Subhi Maraa'beh

Abstract

According to No Free Lunch (NFL) theorem, there is no algorithm that is efficient in solving all optimization problems [54]. Several machine learning (ML) predictive models have been proposed by researchers and these models have showed good results in predicting faulty modules in software projects. However, none of them proved to be good for all software project types. Depending on the dataset characteristics, best machine learning varies from dataset to dataset. In this work, three of the best wrapper feature selection (FS) algorithms (PSO, ACO and GA) are combined together with three machine learning (ML) classifiers (ANN, DT and kNN) to study the best combination out of the nine cross combinations, then a novel approach that combines the results of the nine combinations based on a majority voting technique of the best selected features were introduced and results found to outperform other FS model with significant difference, majority technique increase the average area under curve (AUC) with 0.10 at minimum. This research studied the relation between the search algorithm and the classifier and proposed a new approach of combining the results of the different FS combinations.

الملخص

تبعاً لنظرية لا غذاء مجاني (NFL)، لا توجد خوارزمية واحدة قادرة على حل كل مشاكل التحسين [54]. العديد من نماذج التعلم الآلي المقترحة من الباحثين أظهرت نتائج جيدة في توقع مواقع الأخطاء البرمجية في المشاريع البرمجية. ولكن، لا يوجد نموذج واحد قادر على توقع الأخطاء في جميع المشاريع بنفس الكفاءة. وفقاً لخصائص بيانات المشروع يتم تحديد نموذج التعلم المناسب. في هذا العمل، تم دمج ثلاثة من أفضل خوارزميات اختيار الخصائص (FS) مثل (PSO, ACO and GA) مع ثلاثة من مصنفات التعلم الآلي (ANN, DT and kNN) من أجل دراسة أفضل مزيج من التسعة احتمالات المتاحة، ثم تم طرح نموذج جديد لتوقع الأخطاء في المشاريع البرمجية عن طريق دمج نتائج التسعة نماذج المدروسة، يتم الدمج عن طريق أخذ الخصائص الاكثر استخداما. نموذج المقترح والقائم على دمج النتائج اظهر تحسيناً على مقياس المساحة تحت المنحنى (AUC) بنسبة 0.10 على الأقل. هذا البحث درس العلاقة بين خوارزميات البحث والمصنفات واقترح نموذجاً جديداً لدمج النتائج.

Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

ACO Ant Colony Optimization

AMC Average Method Complexity

ANN Artificial Neural Network

AUC Area Under Curve

CA Afferent Couplings

CAM Cohesion Among Methods of Class

CBM Coupling Between Methods

CBO Coupling Between Object classes

CE Efferent Couplings

DAM Data Access Metric

DIT Depth of Inheritance Tree

DT Decision Tree

FS Feature Selection

GA Genetic Algorithms

IC Inheritance Coupling

KNN k-Nearest Neighbors

L-RNN Layered Recurrent Neural Network

LCOM Lack of Cohesion in Methods

LCOM3 Lack of Cohesion in Methods

LOC Lines Of Code

LR Logistic Regression

MFA Measure of Functional Abstraction

ML Machine Learning

MOA Measure of Aggregation

NFL No Free Lunch

NLM Number of Local Methods

NMC Number of Methods per Class

NOC Number of Children

NOM Number of methods

NPM Number of Public Methods

OO Object Orientated

PSO Particle Swarm Optimization

QA Quality Assurance

RFC Response For a Class

SC Soft Computing

SDLC Software Development Life Cycle

SFP Software Fault Prediction

SLOC Source Lines Of Code

WMC Weighted Methods per Class

Chapter 1

Introduction

Software Quality assurance (QA) is an essential part of developing a robust system that satisfies the users needs and requirements. QA is the set of systematic actions performed by a QA team to provide a sense of trust and confidence that the development life cycle of a software project conforms to established requirements, including functional and technical requirements as well as the managerial requirements of keeping the schedule and operating within budget [30]. QA is a process by itself, it is not a single action that gives a binary output, it is a set of actions and a complex process that looks for any fault inside the system and reproduce it, then report it to the related party to solve, then verify that the solution actually fixed the fault and did not break anything else from system functionalities.

In large scale applications, doing all of the QA activities for every release is a difficult operation in terms of time and budget. Moreover, following the "80:20" rule which states that around 20% of the project is responsible of around 80% of its software faults, identifying error-prone modules will help QA team with targeting the available testing resources on these faulty modules as early as possible [53], which increases the importance of the Software fault prediction (SFP). SFP is

defined as the process/procedure that results in developing models that are used in software development life cycle (SDLC) by software engineers for detecting faulty software modules or classes, the sooner those models are executed the better results we have [53], as a result, better utilization for testing efforts. These models are based on machine learning techniques and/or statistical techniques.

Fault prediction models can be used by different team members; the developers, the QA engineers and the code reviewers during the code inspection session to locate possible faults. Detecting software faults in the early stages of the development life cycle reduces the cost of test efforts and improves software quality [62].

Fault prediction models are totally dependent on finding the suitable software metrics, because of the significant differences in metrics performance. However, this is not an easy task to be accomplished as there are many software metrics with no clear distinction regarding their usability.

1.1 Motivation

Delivering a high quality software is the ultimate goal that any software company is trying to achieve. The definition of high quality softwares varies from company to company based on the software usage, however all of them agree on the importance of avoiding showing bugs, faults or failures to their customers, hence software QA activities are done to maintain the good shape of the product regardless before or after the release of the first product version.

Testing software products using the traditional methods takes a lot of time and does not guarantee good results, since it depends on manual testing done by humans [16]. That's where the importance of SFP arises. Predicting software faulty models helps in finding the faulty modules automatically, targeting

the faulty modules by the QA activities shall reduce the number of faults. The earlier the faulty modules are identified the higher product's quality is gained. SFP models rely on machine learning (ML) and/or software metrics and/or soft computing techniques [63]. SFP models that depends on software metrics use a set of metrics to predict the faults [69], other models depend on the metrics that measure the software change from iteration to another, such models proved to be better predictors [69, 82].

Soft computing (SC) techniques (e.g., machine learning) proved their efficiency in extracting useful information from real life complex systems. The SC techniques are able to extract information from those systems even if the data is incomplete, imprecise, or even incorrect partially. ML algorithms have been used widely in the context of classifying software modules to faulty and none faulty [53]. Such techniques use the software metrics of a software module as features and predict if it is faulty or not based on previous learning experience from similar modules or previous metrics on the same module. Malhotra [53] concluded that ML techniques outperformed the traditional techniques such as the Logistic Regression (LR), which is the most used statistical traditional approach. Artificial Neural Network (ANN) is one of the most used ML technique in predicting the faultiness of software modules in the early stages of developing a software product [55, 28]. However, the performance of ML techniques is highly dependent on the nature of the provided data. Thats to say, the dimensionality of the dataset, the existent of the noisy data, the existence of the redundant and/or irrelevant features. These problems either degrade the performance of the learning algorithm, or increase the required execution time for the learning process. Therefore, data preprocessing is highly required to improve the performance of learning algorithms.

Data preprocessing is an important phase to construct an SFP model. It includes collecting data from multiple sources, converting it to the right format,

clean redundant, noisy and irrelevant data. Data reduction (e.g., Feature Selection (FS)) is one of the most important data preprocessing tasks. According to Liu and Motoda [51], FS aims to reduce the number of features by eliminating non-vital features and keeping the most informative ones. In the case of SFP problem, FS is used to enhance the performance of the SFP model and to reduce the time cost of the learning process [35].

Feature selection is the process of selecting a subset of the most relevant features from the original set of features within predefined threshold [51], FS process consists mainly of two main phase; subset generation, and subset the evaluation. Metaheuristics algorithms proved their efficiency in searching for the most important features in the whole feature set. However, different metaheuristics algorithms show different behavior when dealing with different datasets. In this research, nine combinations of three metaheuristics algorithms (i.e., Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), and Genetic Algorithm (GA)) with three well known classification algorithms (i.e., Decision Tree (DT), k-Nearest Neighbors (KNN), and Artificial Neural Network (ANN)) were proposed as FS algorithms, layered recurrent neural network (L-RNN) was used as validator to validate the features subset generated by the FS combination. Moreover, a novel approach of combining the selected features subsets of each FS combination with a majority voting technique was proposed, and its results were compared with the other nine combinations.

1.1.1 SFP, where to use?

Software quality assurance is a multi-step process that begins from the very early stages of feature development until releasing this feature. SDLC consists of few

phases like planning, analysis, design, implementation and verification [52], however QA is a different concept from verification. QA includes requirements verification, design review, code review, functional testing, non-functional testing, usability and acceptance testing. This research works with SFP models, which allow software engineers to focus development activities on fault-prone code, thereby improving software quality and making better use of resources [35]. Before proceeding with explaining the SFP model in details a clarification where the SFP models can be injected in SDLC is needed, so parties like QA engineers can get insight when to use such models and how can they benefit from their existence.

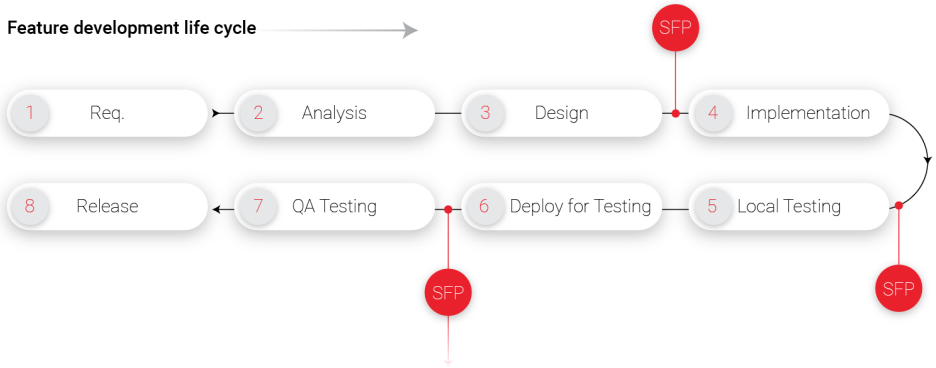

Figure 1.1: Software Fault Prediction model in SDLC

Figure 1.1 clarifies where to inject the SFP model in the SDLC, it is assumed that agile development methodologies have been followed, and iterative development approach is used. In recent years, traditional waterfall methodology has been replaced with agile software development approaches [6]. Examples of Agile methodologies are but not limited to: Extreme programming, Kanban and Scrum [28]. The steps of SDLC are explained as follows:

1. Requirements gathering and analysis

   In this step client needs and requirements are collected, documented and

reviewed to be passed to the development team.

2. Analysis

   In this step the development team start analyzing the requirements document and make sure that they understand what the customer needs.

3. Design

   In this step the development team start designing the feature, and see which software components are to be changed while developing this feature, the result of this step is a design document that explains the effect of adding the new feature on the current systems in terms of which class and/or components to be changed.

   - SFP checkpoint

     After the feature design is ready and the classes/components to affected are known, here where the SFP model can be used to tell us about these classes/components faulty status, are they in a good shape or more effort to be made to solve the issues in these faulty classes, we believe that this step gives the development team better insight on the underlying system healthy status and helps in avoiding costly bugs in early stage.

4. Implementation

   In this step the development team builds the feature, theoretically the development team should leave the code in a better state than before feature development, so again here where SFP model can be beneficial to double check that the project affected classes/components are in a better state than before and not faulty.

- SFP checkpoint

  As mentioned before, after implementing the feature the development team should check that affected classes are still in healthy state if they were healthy, if the to be affected classes were faulty then they should be enhanced to avoid expanding the gap toward bug free modules

5. Local Testing

   In this step the development team performs testing locally to make sure that the feature works as expected and no regression bugs [48], Regression testing is a testing process which is applied after a program is modified.

6. Deploy for Testing In this step the code got deployed to testing servers so the QA team can verify that feature works as expected, as well as no regression bugs has been introduced during the feature development.

   - SFP checkpoint

     SFP model should be run at this stage by the QA team to focus the testing efforts on the faulty modules.

7. QA Testing

   after running the SFP the QA team tests the system using multiple techniques like black box testing, stress testing and load testing to make sure that the system is ready to be released to clients

8. Release

   In this step the new version got deployed to production servers and the clients can see the new system and features that got developed during the development cycle, here where the acceptance testing can be done.

This research studied the performance of three Meta-Heuristic algorithms (e.g., PSO, ACO and GA) when combined with three classifiers (e.g., KNN, DT,

and ANN) to perform as FS methods. Different combinations were investigated to test the assumption that the performance of each model depends on the optimization algorithm and classifier used. Therefore, this research studied the effect of combining different classifiers with different optimization algorithms on the SFP model accuracy.

1.2 Problem Statement

Software projects are getting bigger every day. As a rule of thumb, the more code written the bugs generated. Traditional ways of QA can't find all the bugs and requires long time to test all project aspects [16], therefore new ways for finding bugs has been proposed such as SFP models.

SFP models classify software modules to faulty and non-faulty, so the faulty modules can be targeted with more care by QA team. SFP models depend on software metrics (Features) to classify modules, however, high number of features may drive the model performance back and reduce its accuracy [74]. FS algorithms can be injected to SFP model pipeline to reduce the data dimensionality by excluding irrelevant and uninformative features to enhance the overall model performance and increase its accuracy.

Many FS algorithms are available in the literature, however, according to NFL theorem, there is no superior algorithm that can solve all NP-Hard problems efficiently [78]. Therefore, the current FS techniques can always be improved. In spite of that; the door is still open for more improvements.

In the context of improving the learning outcome by finding the minimal feature subsets from an original data set using meta-heuristic algorithms, this

thesis seeks to answer the following three research questions:

RQ1: Do the FS methods affect the performance of L-RNN model when predicting the software faults?

RQ2: Do the use of different combinations of meta-heuristics algorithms with different classification algorithms, have different affects on the performance of L-RNN?

RQ3: How the selection of the representative features from a dataset affects the performance of the L-RNN model?

1.3 Research Objectives

This research aim is to propose a software fault predictor that is based on different ML techniques, which are enhanced by employing the state of the art feature selection algorithms. Three well-known classifiers (i.e., ANN, DT, and KNN) are used and their efficiency is improved by reducing the dimensionality of the used data samples by employing three bio-inspired feature selection algorithms (i.e., GA, ACO, and PSO). To achieve this goal, three research objectives were formulated as follows:

- To assess the performance of L-RNN classifier on the full datasets of software fault prediction.
- To investigate the influence of different FS methods (nine methods) on the performance of the L-RNN model.
- To assess the role of the mechanism of selecting the representative features (selecting the best feature subset by applying the majority voting mechanism) on the performance of L-RNN classifier.

1.4 Thesis Organization

The rest of this thesis is organized as follows: Chapter 1 includes a general introduction about the main topic of this research, the motivation, the problem statement, research questions and the research objectives. Chapter 2 provides an overview of the SFP models, software quality metrics, ML techniques, feature selection problem and meta-heuristic search algorithms.

In Chapter 3, the most important related works in the context of using swarm-based algorithms, data mining, feature selection, feature selection techniques, SFP models and iterative SFP models and techniques were analyzed and criticized. The research methodology is presented in Chapter 4. The details of running all experiments of the nine FS combinations, and the detailed steps of how to obtain a majority voting solution and apply it in the SFP model are also explained.

Chapter 5 shows the experiments results and discussion of the proposed approaches. The results of running SFP model without FS are discussed in first phase, then the results of running each FS combination alone, then the results of the majority voting mechanism are discussed and compared with the nine FS combinations, in the end the we provide a histogram diagram of the number of selecting each feature in the ten obtained solutions for all the datasets and each dataset alone. Finally, the conclusions and future directions were drawn in Chapter 6. It focuses on general discussion, the main contribution of this research and open the doors for some future works.

Chapter 2

Background

System failure is a condition that causes a system to behave incorrectly in a none expected way, and makes the external system behavior to be incorrect. Usually, systems fail because they do not satisfy the specifications and the user needs, or because the specification may not describe its function. On the other hand, is an internal failure or system state that causes the system failure in case it reached the system interface and affects the users. A fault is a system state that is caused by an error. Its a structural imperfection in a software system that affects the overall system stability and may lead to the overall failure [22].

In the SDLC, faults may begin from the time of requirements elicitation while describing what the user needs from the system, in the design phase, or in the development phase. The cost of fixing faults depends on the time when they were discovered. That's to say, fixing a fault in the requirements phase is much better than discovering it in development phase. Discovering such a fault after delivering the product to the customer, the cost will be much higher [44, 10].

Software quality assurance process can affect each phase of the SDLC. In this section, we explain some of the QA activities [66, 31].

1. Reviews: is a set of review activities held during the SDLC, starting from

requirements until maintenance, usually reviews are made by experts to other team members. Many ways are available to do a review, however all of them aim to catch the faults as early as possible. Moreover, all these ways share that the review highly depends on the reviewer experience and judgment. This activity requires two persons from the development team; the reviewer and the reviewee. Review activities include the following:

   (a) Requirements elicitation reviews

   (b) Formal design reviews

   (c) Code inspections and walk-throughs

2. Expert Opinions: this activity to some extent is similar to the reviews, however, Expert Opinions may include consulting experts from outside the company, to advise on the design or the implementation of some features.

3. Software Testing: many types of software testing are usually applied in the life cycle:

   (a) Black box testing

   (b) White box testing

   (c) Automation testing

   (d) Acceptance testing

   (e) Load testing

4. Software maintenance components:

   (a) Maintenance contract review

   (b) Maintenance plan

   (c) Maintenance staff training

(d) Maintenance procedures

(e) Maintenance quality cost

QA is very important part of the SDLC, below are some reasons [60]:

1. QA is important to check if any errors and defects were generated during the development life cycle.

2. QA provides to the customers the sense of reliability and satisfaction on the application.

3. Quality products bring customer's confidence, It is very important to ensure the Quality of the product.

4. Testing results in a higher quality products that require lower maintenance cost, and gives more accurate, reliable and consistent results.

5. Testing will ensure satisfying system non-functional requirements, testing will check some intangible important aspects of software such as system performance and throughput.

6. It's important to check that the system does not have any failures and will not crash as this will be very expensive in the future or in the later stages of the development [61].

7. High quality products are required for any software organizations to stay in business.

2.1 Software Fault Prediction (SFP)

SFP is the process/procedure that produces models that are used in the SDLC for predicting the faulty software modules or classes, the sooner those models are

run the better results we have [53], as a result better utilization for testing efforts [36]. These models are built using machine learning techniques and/or statistical techniques.

SFP classifies software project modules to faulty and non-faulty by building a model on fault data taken from similar projects or from previous releases of the same project, if the module contains faults then its value is recorded as 1 (or 0 if does not). For SFP models software metrics are independent variables and the faulty instances in the underlying dataset are dependent variables [13].

Below are the benefits of using those SFP models in development life cycle [14, 15]:

- Enhance system the overall code quality by pointing out modules/classes that need refactoring, and direct the developers eyes on them.
- Gives the alerts for software testing team to focus more on faulty modules to test which will spare their time from checking and testing healthy modules.
- Enhance the software design by providing more decision points by identifying the fault-prone classes by using class-level metrics.
- Fault prediction is an approach to reach dependable systems, using those models will improve the system stability and reliability.

2.2 Quality Metrics

Software metrics are widely accepted tools to control and assure the software quality [64]. There are various software metrics of content can be used in SFP. The most popular types of metrics for software prediction are listed below (see Figure (2.1)):

1. Procedural Metrics:

   metrics that measure the cost of a software project, or of some project activities such as documentation, original development, software maintenance. Procedural metrics also cover effort metrics estimating the human part, required for the completion of a product under construction. Earlier development procedural metrics can be reused to assessing how much of the development would be required for future projects, other metrics like lines of code (LOC) considered to be procedural.

2. Object Oriented Metrics:

   Object Oriented (OO) metrics play an essential role in the developing a bug and fault free software systems. Object Oriented is famous because that it focuses on objects as the prime agents involved in the computation to reflect real life objects, each system entity will represent a class of data and the related operations, object oriented design will give a re-usable, modular and tidy code. Object oriented metrics are used to measure properties of object oriented designs, metrics such as cohesion, coupling and inheritance for a class. In some reference they even consider Lines Of Code (LOC) and Source Lines Of Code (SLOC) as object oriented metrics.

3. Process Metrics:

   Process Matrices include code changes (delta) metrics, process, code churn, history and developer metrics. These software metrics can be obtained from the source code and the code repository [62]. These metrics are collected iteratively in the course of system releases.

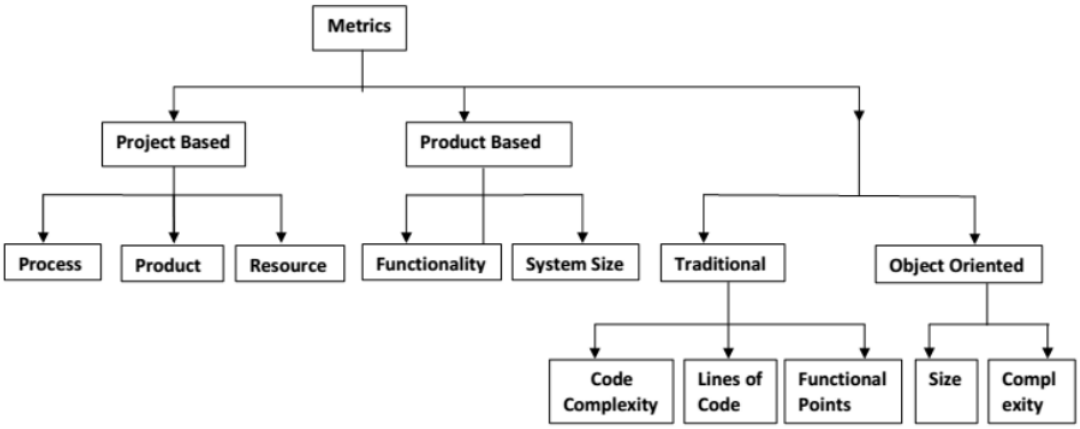

Figure 2.1: Quality Metrics

The above mentioned software metrics have been widely used in literature in different SFP models. Based on [62], almost 49% of the studies used OO metrics in SFP models, and around 27% of the studies used traditional metrics and in the end 24% of the studies used process metrics.

Which metrics are useful for SFP?

- LOC (lines of code): found to be useful and used till these days in research.
- McCabe's cyclomatic complexity: found to be useful and extensively used, according to [62] this metric found to be more effective in large scale projects in the context of SFP.
- Simple WMC (weighted methods per class): The complexity of local methods in the given class, The WMC is the number of local methods in the given class.
- NLM (Number of local Methods): is the number of local methods in a class that can be accessed outside the class -> public methods.

- NOM (Number of methods): is the number of all methods in the class including the inherited ones.
- NMC (Number of methods per class): the number of public, private and protected methods in a class not including the inherited ones.
- WMC LOC: is a metric per method per class the method complexity is defined using LOC metric. The WMC is equal to the sum of lines of code of all local methods in a class. In this case, WMC is a size measure.
- WMC McCabe: is a metric per method per class, the complexity of each method is calculated using McCabe's cyclomatic complexity. The WMC is the sum of all local methods McCabe's cyclomatic complexity in a class. In this case, the WMC is a complexity measure.
- DIT (Depth of Inheritance Tree): measures the number of ancestors of a class [34].
- NOC (Number of Children): Number of immediate descendants of a class [34].
- CBO (Coupling Between Object classes): the coupling between two classes A and B if class A uses a method and/or instance variable of class B. The CBO gives the number of classes to which a given class is coupled.
- RFC (Response For a Class): Count the number of distinct methods invoke by a class in response to a received message.
- LCOM (Lack of Cohesion in Methods): Count the number of methods do not share a field to the method pairs that do.
- CA (Afferent Couplings):is a metric to measure coupling by measuring how many other classes use the class.

- CE (Efferent couplings): Count the number of classes to which a class depends.
- NPM (Number of Public Methods): Number of public methods defined in a class.
- LCOM3 (Lack of Cohesion in Methods): Count the number of connected components in a method graph.
- DAM (Data Access Metric): Computes the ratio of private attributes in a class.
- MOA (Measure of Aggregation): Count the number of data members declared as class type.
- MFA (Measure of Functional Abstraction): Shows the fraction of the methods inherited by a class to the methods accessible by the functions defined in the class.
- CAM (Cohesion Among Methods of Class): Computes the cohesion among methods of a class based on the parameters list.
- IC (Inheritance Coupling): Count the number of coupled ancestor classes of a class.
- CBM (Coupling Between Methods): Count the number of new or re-defined methods those are coupled with the inherited methods.
- AMC (Average Method Complexity): Measures the average method size for each class.
- max_cc (McCabe's cyclomatic complexity): counts the maximum number of logically independent paths in a method.

- avg_cc (McCabe's cyclomatic complexity): counts the average number of logically independent paths in a method.

In general, Object Oriented and process metrics are useful for SFP, as for the object oriented metrics the most successful metrics are the CK metrics [20, 19]. The CBO, WMC and RFC are considered as the best from the CK metrics suite [62]. The code coupling metrics are more useful than inheritance and cohesion metrics [62].

It is important to define two concepts, pre and post release faults, pre-release faults are the faults that appear before releasing the first version of the system, post-release faults are the faults which are shipped and released, the post-release faults are more vague and harder to predict.

Process Metrics are mainly related with post-release metrics such as Delta metrics, delta metrics are calculated as the difference of metrics values between two versions of a software, process metrics are better than OO and traditional metrics when it comes to SFP.

2.3 Machine Learning (ML)

ML is the process of enhancing the performance of computer program in a progressive manner. One of the most important ML applications is classification. Classification is the process of mapping an input sample into one of the categories of the original data based on a set of features.

The main idea behind the classification algorithms is to build a model that can classify unseen inputs. Spam filtering is a common problem that has been tackled using classification algorithms.

ML is multi-step process, the data need to be collected, then data need to be pre-processed, if we know which features are the most informative ones then we

can use them otherwise feature reduction techniques need to be applied to enhance the classification performance and accuracy. Feature reduction techniques are defined as the set of processes that aims to remove irrelevant and noisy features from a given dataset to keep the best fit subset of features. Data reduction helps not only in enhancing the performance of the classifier also in enhancing its accuracy by removing noisy features. Classifier accuracy can be calculated in different ways, one way is to split the data set into training set used to train the model and evaluation set used to calculate the performance of the model, by counting the number of correctly and incorrectly classified test records, those numbers are explained in the confusion matrix [29].

Table 2.1 explains the confusion matrix for binary class problem, the records in the below table are explained as follows, TP is the number of records that belong to class A and classified as class A, TN is the number of records that does not belong to class A and classified as not class A, FP is the number of records that does not belong to class A and classified as class A, FN is the number of records that belong to class A and classified as not class A, generally speaking TP and TN are the number of correctly classified records, However, FP and FN are the number of incorrectly classified records.

		Predicted Class	
		Class A	Not Class A
Actual Class	Class A	TP	FN
	Not Class A	FP	TN

Table 2.1: Confusion Matrix for binary class problem

The most popular metrics to evaluate a ML model are area under curve (AUC), accuracy and error rate [74, 29, 33], which are defined in equations 2.1 and 2.2:

$$Accuracy = \frac{TP + TN}{Total} \tag{2.1}$$

$$ErrorRate = \frac{FP + FN}{Total} \tag{2.2}$$

Below explained some of learning methods to be used in this research.

1. Decision Tree [29] is a simple commonly used classifier in data mining which creates a model that predicts the value of the target feature (class) for the test sample based on the values of the input features vector. In the tree each non-leaf node corresponds to one of the input features. There are edges from a node to its children for each the possible value for that feature, and each leaf node represents a decision point (class label), each path from root to leaf represents a classification rule, an example of simple decision tree illustrated in figure 2.2 below.

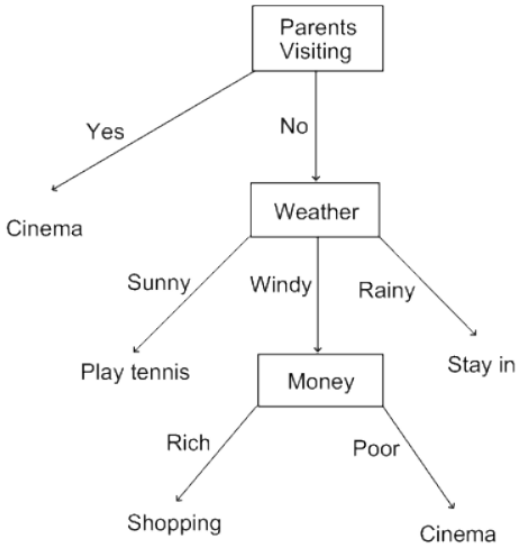

Figure 2.2: Decision Tree.

Decision tree classifier is an eager learner [74]. There are many ways to construct a decision tree from a given sample data, those trees different from each other, however finding the optimal tree is an expensive operation. Many algorithms have been developed to help find a near optimal tree, those algorithms use a greedy strategy to grow the decision tree in a top-down approach, one of the most famous algorithms is Hunt's Algorithm, Hunt's Algorithm forms the basis of many implementations of DTs such as ID3, C4.5, and CART.

To construct a DT an attribute must be selected as the root node. To get the smallest and most efficient tree the root node must effectively split the data, each split will try to classify/label a subset of the instances, the most efficient split is the one that labels the highest number of instances which is referred to as the most information gain [42]. The splitting continues until all the instances are label/classified, DT is simple classifier and easy

to understand as it can be visualized as graph.

2. Artificial Neural Networks (ANN)[1] is a very interesting topic in the research field and in the study field. Neural Networks consists of huge number of connected processors called neurons. The ANN has the capability to develop an internal representation of a signal pattern that is presented as input to the network [37].

   ANN is widely used as a classifier for solving the classification problems. ANN is a ML algorithm that need to be trained, once trained ANN can predict the classification of new data. ANN is able to function and learn even if the data is noisy. ANN has many models starting from the simplest which is the perceptron (one layer neural network) to multi layer neural networks, feed forward and recurrent networks.

   Perceptron model consists of input nodes, weighted links and output nodes, let's assume that we have a function that receives three boolean inputs and outputs one if at least two of the input are true and outputs minus one otherwise, table 2.2 illustrates the data for the example.

X1	X2	X3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	0	-1
0	0	1	-1
0	1	0	-1
0	1	1	1

Table 2.2: Perceptron data set example

The perecptron model for the above dataset consist of three input nodes, three weighted links and one output node, one solution is

$$\hat{Y} = \begin{cases} 1, & if \quad 0.3X1 \ + \ 0.3X2 \ + \ 0.3X3 \ - \ 0.4 \quad > \ 0; \\ -1, & if \quad 0.3X1 \ + \ 0.3X2 \ + \ 0.3X3 \ - \ 0.4 \quad < \ 0; \end{cases} \tag{2.3}$$

Where the 0.3 is the weight to learned and adjusted by the perceptron and the 0.4 is bias factor t, the above equation can be seen as

$$\hat{Y} = sign(w_d x_d + w_{d-1} x_{d-1} + ... + w_1 x_1 + w_0 x_0 - t) \tag{2.4}$$

The perceptorn keep updating the weights until a stopping condition is met based on the following equation

$${w_j}^{(k+1)} = {w_j}^{(k)} + \lambda(y_i - \hat{y}_i^{(k)}) x_{ij} \tag{2.5}$$

Where ${w_j}^{(k)}$ is the current weight, λ is the learning parameter that varies from 0 to 1 and x_{ij} is the value of the j^{th} attribute in the training example. λ value need to be set wisely low values will make the new weight very influenced by the old weight, however, high λ values will make the new weight very influenced by the change of each new iteration, usually λ values starts with relatively hight values and keep decreasing in each iteration [74].

The perceptron model described in equation 2.4 is linear combination of x and w, and not all the spaces are linearly separable, more complicated functions can be introduced to solve this problem.

3. Layered Recurrent Neural Networks (L-RNN): is more complex than the perceptron model, it can contain multiple layers of perceptrons with hidden layers, and can use other activation functions than the sign function such as sigmoid, linear and hyperbolic tangent functions. L-RNN can solve any classification problem including non-linear spaces [74]. The learning process of L-RNN is a time-varying pattern, applying either feed-forward or feedback connections.

   Training L-RNN is similar to training the ANN, however, with a small tweak. Each output node value depends on the current step calculations and on the previous time step calculations not only on the current step calculations like ANN. In L-RNN the output of some nodes is an input for other nodes so it works as a feedback channel for the network. Feedback nodes remember the values of the previous stage, so the output data of each stage will depend on the input data of the current and previous stage [49].

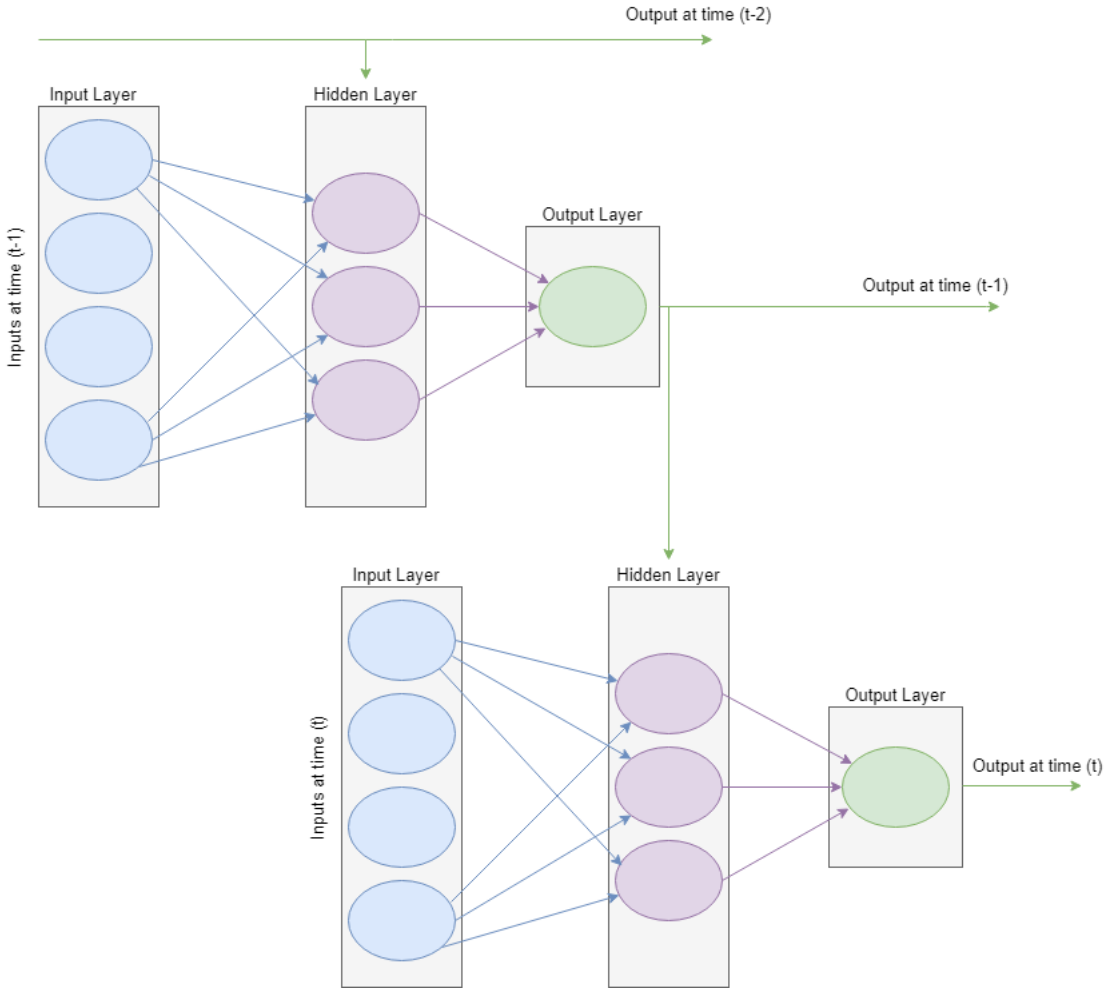

Figure 2.3: AN example of L-RNN.

Figure 2.3 shows an example of L-RNN at time t. Given an input vector I=(I_1, ..., I_t), L-RNN computes the hidden vector H=(H_1, ..., H_t) and output vector O=(O_1, ..., O_t) by multiple iterations using equations 2.6 and 2.7.

$$H_t = f(W_{HI}I_t + W_{HH}H_{t-1} + b_H) \tag{2.6}$$

$$O_t = f(W_{OH}H_t + b_O) \tag{2.7}$$

Where f() is the activation function (i.e. sigmoid, linear or hyperbolic tangent functions). W_{HI}, W_{HH} and W_{OH} are weight matrices as follows: (i) W_{HI} matrix that shows the weights between input layer and hidden layer, (ii) W_{HH} matrix that the weights between a hidden layer with itself at certain time slot and (iii) W_{OH} matrix that shows the weights between a hidden layer and output layer. b_H and b_O are vectors that present bias parameters which help each recurrent neuron to learn an offset.

4. kNN (k nearest neighbors) [74] is a simple straight forward classification technique and one of the most widely used learning algorithms. From its name kNN depends on the majority vote of the k nearest neighbors to classify the new record, kNN is a lazy non-parametric algorithm. That means that it does not require any prior knowledge about the underlying data. Therefore, kNN could be one of the choices for classification when there is a little or no prior knowledge of the data distribution. The basic idea is "If it walks like a duck, quacks like a duck, looks like a duck, then it's probably a duck.".

   kNN needs three things to operate: 1- a training set of labeled records, 2- Distance metric to compute the distance between the given records such as Euclidean, cosine, Chebyshev distance [74], 3- k value which is the number of neighbors to be considered to label the new record. figure 2.4 illustrates the definition of k nearest neighbors, k value need to be set wisely as small k value makes the model sensitive to noise points, however, large k values may return invalid results as it contains values from other classes.

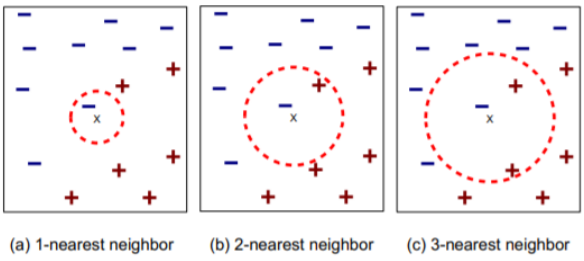

Figure 2.4: Definition of Nearest Neighbors.

2.4 Feature Selection (FS)

FS [59], is the process of deciding on a subset of important features for building reliable learning models. Feature selection methods are classified as Filters or Wrappers according on how they evaluate the feature subset [50].

Filter methods select the set of features regardless the model itself, they use some known metrics to decide which features should be eliminated; like Correlation, Information Gain and entropy [45] [50]. Filters methods might select redundant variables since they do not consider the relation between the variables. While in Wrappers method; they try to select subset of variables to learn the model. Adding or removing features from large datasets decision based on the resulting accuracy. Usually these methods become searching methods due to their high computational cost [80].

Based on No-Free-Lunch (NFL) theorem, there is no machine learning algorithm that is best for all software projects fault prediction. Each project has its characteristics that makes a certain algorithm is best to predict its faults, so instead of working on a one-size-fits-all algorithm Dores, Alves and Ruiz switched to study a machine learning recommendations. They introduced a novel framework for recommending machine learning algorithms that is capable of automatically identifying the most suitable algorithm according to the software project for fault

prediction [22].

Any FS algorithm can be seen as mixture of two steps, first search strategy that aims to select a features subset from the original features vector, and the evaluation step that evaluates the accuracy of the selected subset and evaluate the goodness of the first step [51].

FS process can be visualized as three dimensional space of Evaluation measure (Accuracy, Consistency and Classic), Search Strategy (Complete, Heuristic, Non-Deterministic) and Generation Scheme (Forward, Backward and Random).

FS models can be multivariate and univariate based on the number of features to be added to the current solution, univariate add/remove only one feature at a time from current features subset, however, multivariate add/remove more than one feature at a time.

Filters. Filter methods filter out features based on some evaluation criteria independently of the learning algorithms. Evaluation criteria such as features correlation and Shannons Entropy are widely used evaluators [33] figure 2.5 illustrates the process of filter feature selection methods.

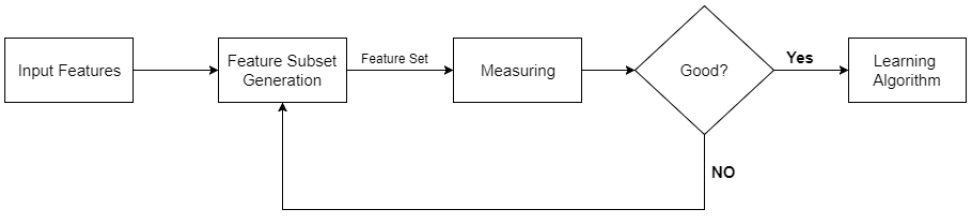

Figure 2.5: Filter FS Methods.

Wrappers. As the feature selection process is pre-processing phase to enhance the performance and accuracy for a learning algorithm then the learning results should be taken into consideration in feature selection process [39].

Instead of using mathematical metrics as in filter methods Wrappers use the learning algorithm metrics such as Accuracy, error rate and AUC (Area Under

Curve) to evaluate the selected features [51, 39, 17].

Wrappers are more time consuming than Filters as the learning algorithm/Classifier (e.g kNN, DT, ANN) should be run as the subset evaluator in each iteration to determine the goodness of the given subset [71, 51], The good news are despite of this performance drawback Wrappers leads to a better classification accuracy than Filters [71]. Figure 2.6 illustrates the process of wrapper feature selection methods.

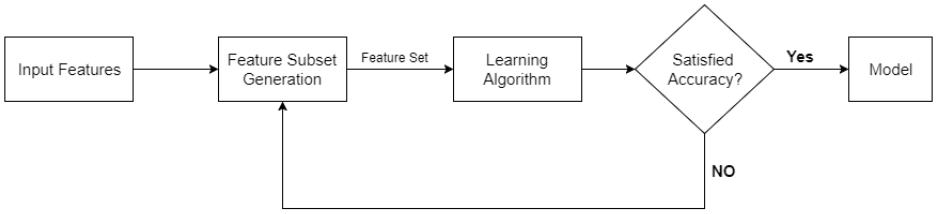

Figure 2.6: Wrapper FS Methods.

2.5 Meta-heuristic Optimization Algorithms

Feature selection is NP-hard problem [45], so finding the ultimate set of features that gives the best accuracy can't be obtained without searching the whole solution space. Assuming that n features are available in a dataset, then there will be 2^N solutions [33]. The use of Meta-heuristic optimization search algorithms helps in finding near optimal solution without the need to search the whole space of solutions [45] [73]. In order for feature selection model to execute effectively, it must be combined with a good search strategies. Subsets can be generated in different ways for example starting with empty set of features then start adding features to the empty solution, this strategy called sequential forward search (SFS), another way is to start with a full set of features and start removing features from that set, this strategy called sequential backward search (SBS).

SBS and SFS suffer from the fact that the feature is added or removed cannot be added or removed again. That will reduce the chances of getting the optimal solution. Random subset generation strategy can solve the mentioned problem. However, the cost again is too high.

The strategies discussed above perform local search rather than global search, so the chance of getting the optimal solution is low, beside the problem of getting stuck in local optima.

Meta-heuristic algorithms tends to execute global search and local search in a mixed manner, the idea is to utilize multiple search agents each initialized to search in different area in the search space, each agent will start searching the solution space while balancing between Exploration and Exploitation criteria.

Exploration and Exploitation are two contradictory criteria that should be considered when using metaheuristic search algorithms [73]. The metaheuristic algorithms are classified based on these two criteria into two categories; population-based algorithms like Swarm intelligence, and local-search based algorithms like simulated annealing. Each of which has advantages and disadvantages, so finding a hybrid algorithm that combine both criteria exploration and exploitation will result in good FS searching results. The resulting combined algorithm is called memetic algorithm [72].

Below introduced some of the algorithms that will be used in this research.

1. Particle Swarm Optimization PSO is a population based stochastic optimization technique developed by Eberhart and Kennedy in 1995 [7]. The algorithm is a mimic of flock of bird movement, it comes in both discrete and continuous versions. FS can be seen as a discrete problem.

   PSO is a population global search technique, where a set of particles are

defined. Each particle represents a potential solution. The particles initialized with a random solution inside the solution space, then each particle tends to optimize its solution based on some fitness function. The velocity of each particle is a component of a random solution, personal best solution and global best solution.

Personal best solution is the solution that gives the highest fitness value of the particle itself so far. Global best is the best solution obtained of the whole system. The particle velocity updated based on equation 2.8 and its position is updated based on equation 2.9.

$$v_{id}(t+1) = w(t)v_{id}(t) + c_1 r_{1d} * (X_{pbest}(t) - X_{id}(t)) + c_2 r_{2d}(X_{gbest} - X_{id}(t)). \tag{2.8}$$

$$X_{id}(t+1) = X_{id}(t) + v_{id}(t+1) \tag{2.9}$$

Where w is inertia weight usually a positive number, r_1 and r_2 are two randomly generated number from 0 to 1, and c_1 and c_2 are other two variables to control the Exploration and Exploitation of search criteria, Algorithm 1 describes the pseudo code of PSO.

Algorithm 1 PSO Algorithm

```
Parameters used for PSO in this research
let swarmSize: 40
let numberOfIterations: 3000
let initialVelocity: initial velocity
let initialPosition: initial position
let c1: 1.5 //variable to control the influence by global best
let c2: 1.5 //variable to control the influence by global best
let w: 0.8
initialize particles()
while currentIteration <= numberOfIterations do
    for Each particle i do
        update the velocity of each particle based on equation 2.8.
        update the position of each particle based on equation 2.9.
        if new personal best is better than old personal best then
            update the personal best solution.
        end if
        if personal best is better than global best then
            update the global best solution.
        end if
    end for
end while
return the global best solution
```

Figure 2.7 simulates how the particles initially may be distributed on the search spaces and how particles converges to a certain solution in the search space after keep updating their velocity in every iteration.

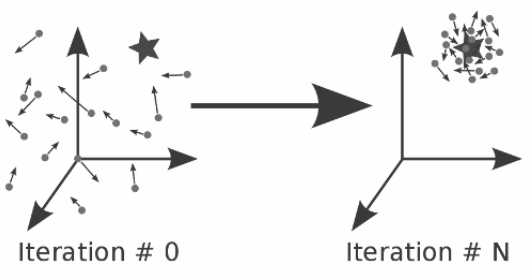

Figure 2.7: PSO particles simulation.

2. ACO Ant colony optimization (ACO) [23] is a population-based metaheuristic algorithm for solving optimizing problems. Initially proposed by Marco Dorigo in 1992. The algorithm mimics the real behavior of ants to find the shortest path, each ant will follow some indications left by other ants called pheromone. Each individual ANT of the population will build the solution for the given problem incrementally and stochastically.

   Optimization problems needs to be represented in a graph based representation so the ants can build its solutions. At each iteration ants moves adds or removes solution components so the final solution is ready. ACO [41, 24, 26] is another promising approach to solve the combinational optimization problems and has been widely employed in FS [2, 4].

   The first usage for ACO was to solve the famous Traveling Salesman Problem (TSP) [25] and then to many other NP-hard problems such as vehicle routing, Quadratic assignment Problem (QaP), scheduling and system fault detection [9]. Two terms need to be defined: "update selection measure (USM)" and "local importance (LI)" in the aco-based FS method. ACO similar to PSO starts with initializing set of ants in random positions in the search space, then each ant will have a probability to update the next move.

Algorithm 2 ACO Algorithm

Parameters used for ACO in this research

let numberOfAnts: 20

let numberOfIterations: 3000

let initialPheromon: 1

let α: 0.8

let β: 0.8

let evaporationRate: 0.6

Evaluate initial population according to the fitness function.

Find the best solution of the population.

while $currentIteration <= numberOfIterations$ do

Do until each ant build a solution

local trial update.

End Do

Update the pheromone.

Determine the best global ant.

end while

return the global best solution

3. Genetic algorithms (GA) [32] are heuristic learning models based on principles taken from natural selection. In GA [27], three main genetic operators (selection, crossover, and mutation) can be implemented in different ways. GA starts with generating set of solutions as the initial set, then it applies the genetic operators on those solutions iteratively until stop critera got satisfied. For example number of iterations or accepted solution has been generated.

   The effect of applying GA crossover is huge on the solution, so that will

move the affected chromosome a large distance in problem space. Iteration after iteration all the solutions will move toward a similar structure and the crossover will have less effect on the affected chromosome (solution). So the effect of GA operations will start with huge impact and decay over time.

Algorithm 3 Genetic Algorithm

Parameters used for GA in this research

let populationSize: 40

let numberOfIterations: 3000

let crossoverRate : 0.7

let mutationRate: 0.1

let selectionType: Roulette Wheel Selection

let corssoverType: single, double or uniform

Generate initial population.

Evaluate initial population according to the fitness function.

while $currentIteration <= numberOfIterations$ do

Breed crossoverRate x populationSize new solution.

Select two parent solutions from current population.

if $rand(0, 1.0) < mutationRate$ then

Mutate the child solution.

end if

Evaluate the child solution according to the fitness function

Add child to population.

Remove the crossoverRate x populationSize least-fit solutions from population.

end while

return the global best solution

Chapter 3

Related Work

Software projects and components are getting more sophisticated and dependent on other projects. Therefore, high quality and easy-to-maintain software became a harder mission, keeping in mind that cost shouldn't be ignored.

Software engineering is significant and fundamental in order to mend software quality and minimize maintenance efforts before deploying systems. Software engineering includes varied prediction processes like test effort prediction, correction cost prediction, reusability prediction, fault prediction, quality prediction and security prediction.

Many of these processes are in their early stages and further studies are required to reach stabilized and powerful models. Software fault prediction is the current research area of these prediction processes and lately multiple research centers are working on new projects.

SFP models have been studied from 1990s till today, where fault-prone modules can be specified before system tests using these models. Based on recent studies, the detection probability (PD) (71%) of fault prediction models possibly is higher than PD of software reviews (60%) if a solid model is built [57].

In literature, different models are suggested to tackle the problem of SFP, this

includes the usage of DT, ANN and kNN as classifiers as well as the usage of different optimization mechanisms for feature selection and data lessening which includes ACO, PSO and GA algorithms.

Mandelbugs are faults that are generated under complex conditions, such as integration bugs when interacting with other systems software or hardware, or events ordering. Carroza et al. [12], introduced a new set of metrics that they claimed to better predict Mandlebugs over traditional metrics. Those metrics are related to Concurrency, I/O and Exception handling.

Carroza et al. worked on five prediction models (SVM, DT, MLP, BNs and NB), the results showed that MLP and SVM outperformed other approaches, the experiments were performed on NASA repository datasets, the evaluation has been performed using the k-fold cross validation.

Cahill, Hogan, and Thomas [11] introduced new classification method called Rank Sum to classify faulty software modules from other non-faulty modules. They compared the performance of this new approach with studies based on the Support Vector Machine (SVM) and Naïve Bayes (NB) Classifiers and evaluated using the NASA Metrics Data Program (MDP) data sets.

Erturk and Sezer [28] proposed an SFP system that works on two stages. At the first stage Fuzzy Inference System (FIS) was used without prior knowledge about the project. In the second stage where some data about the project is known (later development iterations) they applied ANN. Their model that follows an iterative approach was better than other models. The authors took one step forward and said that their model can predict software faults online. Moreover the authors implemented a plugin on eclipse of their work.

Shatnawi [67] used ROC analysis. His results showed that those four metrics (WMC, CBO, RFC and LCOM) has significant relationship with faults generated on most of the releases. Shatnawi analysis applied on five systems, also he applied

ROC analysis to include/exclude features. In other words ROC analysis was used as FS method. Shatnawi solution was validated using four ML models, logistic regression, naïve Bayes, the nearest neighbors and C4.5 decision trees. His results does not show a major improvement when used ROC analysis as FS.

Kalsoom [40] discussed the effect of two problems that face the ML learning classifiers. Which are the imbalance problem and irrelevant features. Imbalance problem is caused by redundant instances of similar classes, sampling or reduction of class instances handles this problem. The research used Fisher linear discriminant analysis (FLDA) as FS method. SMOTE and Resample used as sampling strategies, they evaluated their results using nine classifiers using Precision, recall, f -measure, and AUC as performance measures.

Malhotra et al. [53] in her systematic review showed that ML techniques can be very helpful in SFP. However, the usage of those ML techniques is still not widely used and sophisticated. More research need to be conducted to get more general and applicable results. Her results also showed that DT was better than other ML algorithms and Linear Regression models.

Moreover, Malhotra analyzed what software metrics are found to be useful for SFP and found that CBO, RFC and LOC are the most useful among OO metrics. As for the performance measures; Accuracy, precision, Recall, AUC and F-Measure are the most used measures. The average AUC values for predicting faulty modules varies from 0.7 to 0.83. She also observed that the evolutionary algorithms such as ACO are not widely used in the SFP domain, noting that this research used ACO as one of the FS algorithms.

Menzies et al. [57] after comparing different ML techniques concluded that the best software quality metrics (features) to be used in SFP models are different based on the dataset characteristics, and so does the algorithms performance. The experiments has been carried out over public domain datasets.

Lessmann et al. [47] worked on the idea of comparing different ML algorithms in the context of SFP. They built a framework to do this. they concluded that no clear performance differences between different ML algorithms. Results were built upon running 22 distinct classification model on 10 public domain datasets[22].

Song et al. [70] recapped that no studied schema takes control where the solution is to select assorted schema according to the selected dataset characteristics.

Tosun, Turhan, and Bener [75] drove tests on public datasets to prove the validity of Zimmermann and Nagappan's paper published in ICSE'08 [82]. Three embedded software projects were used for the analysis. They concluded that network measures are important indexes of fault-prone modules for large systems. Performance valuation metrics used were PF, PD, and precision.

Chang, Chu, and Yeh [18] suggested a fault prediction process according to organization rules to find out fault paradigms. They declared that prediction results were stellar. The advantage of this method is the detected fault paradigms that can be used in causative analysis to detect the sources of faults.

Mende and Koschke [56] assessed lines of code metric based prediction on thirteen NASA datasets and this model acted well in conditions of area under ROC curve (AUC) parameter and they weren't able to show statistical variations to some data mining algorithms. When effort-sensitive performance measure was applied, a line of code metric based prediction process was the worst.

Binkley, Feild, and Lawrie [8] applied linear mixed-effects recession model on Mozilla by using QALP score and total lines of code except blank lines and comments. Determination coefficient was applied as performance valuation parameter. They declared that neither size measure nor QALP is good predictor for Mozilla project.

Arisholma, Briand, and Johannessen [5] assessed fault-proneness models on a large Java legacy system project. They concluded that modeling mechanism

has limited impact on the prognosis accuracy, process metrics are very useful for fault prediction, and the best model is very dependent on the performance valuation parameter. They suggested a surrogate measure of cost-effectiveness for models assessment. Adaboost combined with C4.5 provided the best results and mechanisms were applied with default parameters.

Shatnawi and Li [5] investigated three iterative release of eclipse. The aim was to study the effectiveness of software quality metrics in SFP for post-release software defects. The results showed that the prediction accuracy decayed release after release. They used AUC as Performance evaluation metric. They performed there experiments on class level metrics and they managed to predict error prone classes and error severity after each release.

One important take out was that error severity is more useful than faulty class classification, since saying that this class is faulty may give false alarm on very low-severity faults.

Bingbing, Qian, Shengyong, and Ping [79] compared k-means clustering method with Affinity Propagation clustering algorithm on two datasets. Their results showed that Affinity Propagation was better than K-means clustering on those datasets following Type-II error. They used Type-II error, Type-I error, and entire correct classification rate (CCR) as performance evaluation metrics. The two datasets that been used are Celestial Spectrum Analysis System and medical imaging system datasets. Affinity Propagation reduced Type-II error and increased CCR on software datasets and the number of cluster was two.

In recent years new trend to Ensemble methods for binary classification has occurred and used widely. Misirli et al. [58] presented an ensemble method for SFP by combining three different techniques namely Naïve Bayes, ANN, and Voting Feature Intervals. The results showed that ensemble classifier outperformed Naïve Bayes classifier in terms of prediction accuracy.

Twala [77] used five SFP approaches as base learners for ensemble method. The results again showed that ensemble methods outperformed individual classifiers. The experiment were performed on a large space system.

Other studies [3, 43] have introduced new ensemble methods for SFP and again emphasize on the above results that ensemble methods out perform the individual classifiers.

Based on this review, it can be concluded that the method level metrics were mostly used in predicting faulty modules in software components. Moreover, ML mechanisms have been widely applied in SFP systems, and showed better performance than that achieved by the traditional statistical methods. However, the automatic prediction of the faulty parts in software components is not popular in industry yet, especially with the emergent of automatic most of the real life processes.

Moreover, this research proposes a study of different search algorithms and classifiers combinations to check the correlation and affinity of search algorithm and the classifier used. Also a new way of merging the results of different models using majority voting was proposed, this shall help researchers and software engineers to save time and efforts, as well as to achieve high-quality results for testing software.

Chapter 4

Research Methodology

This research studies the performance of different FS methods that combine a set of SI algorithms(i.e., PSO, ACO, and GA), with different ML algorithms (i.e., KNN, DT, and ANN), where nine combinations of these algorithms have been tested and evaluated. Figure 4.2 describes the proposed model.

Due to the stochastic nature of the SI algorithms, and due to the fact the each algorithm has different behavior from other algorithms, three different SI algorithms were adopted in this research. Moreover, three well-known classification techniques were used as evaluators in a wrapper FS model. Nine combinations of the SI (Swarm Intelligence) algorithms with the classification techniques were tested, and the produced feature subset is recorded.

4.1 Research Approach

This thesis approach runs in a pipeline starting from Data collection where datasets are collected and prepared for later stages ending with results validation where FS solutions are validated using L-RNN classifier.

4.1.1 Data Collection

In this research nineteen software fault project were used from PROMISE [65] public software engineering repository. These datasets are noise free and have no missing values.

4.1.2 Implementation and validation Phase

Three SI algorithms and three classifiers were adopted in this thesis, code prepared on Matlab environment, SI algorithms code has been taken from Yarpiz code store, and for classifiers Matlab native libraries has been used.
Results clustered to three clusters based on FS type. First without FS where no FS algorithm has been used, Second with FS running the 9 different FS combinations, Third with Majority FS model. Results without FS model are generated by sending each dataset as is without reducing its dimensionality to L-RNN classifier (evaluation phase). Avg. AUC value has been saved for each dataset after 100 run per dataset.

Figure 4.1 shows the process followed in this thesis to generate results with FS model. For each dataset run all SI and classifiers combinations, for each combination keep iterating until a stop criteria has been met, stop criteria is number of iterations or threshold fitness value has been reached.

By that the best solution was found. This solution is used to reduce the dataset based on the selected features. The reduced dataset separated into training and testing datasets, training dataset trained the L-RNN evaluator and testing dataset evaluated the accuracy of the solution. In the end L-RNN model and AUC got saved for analysis.

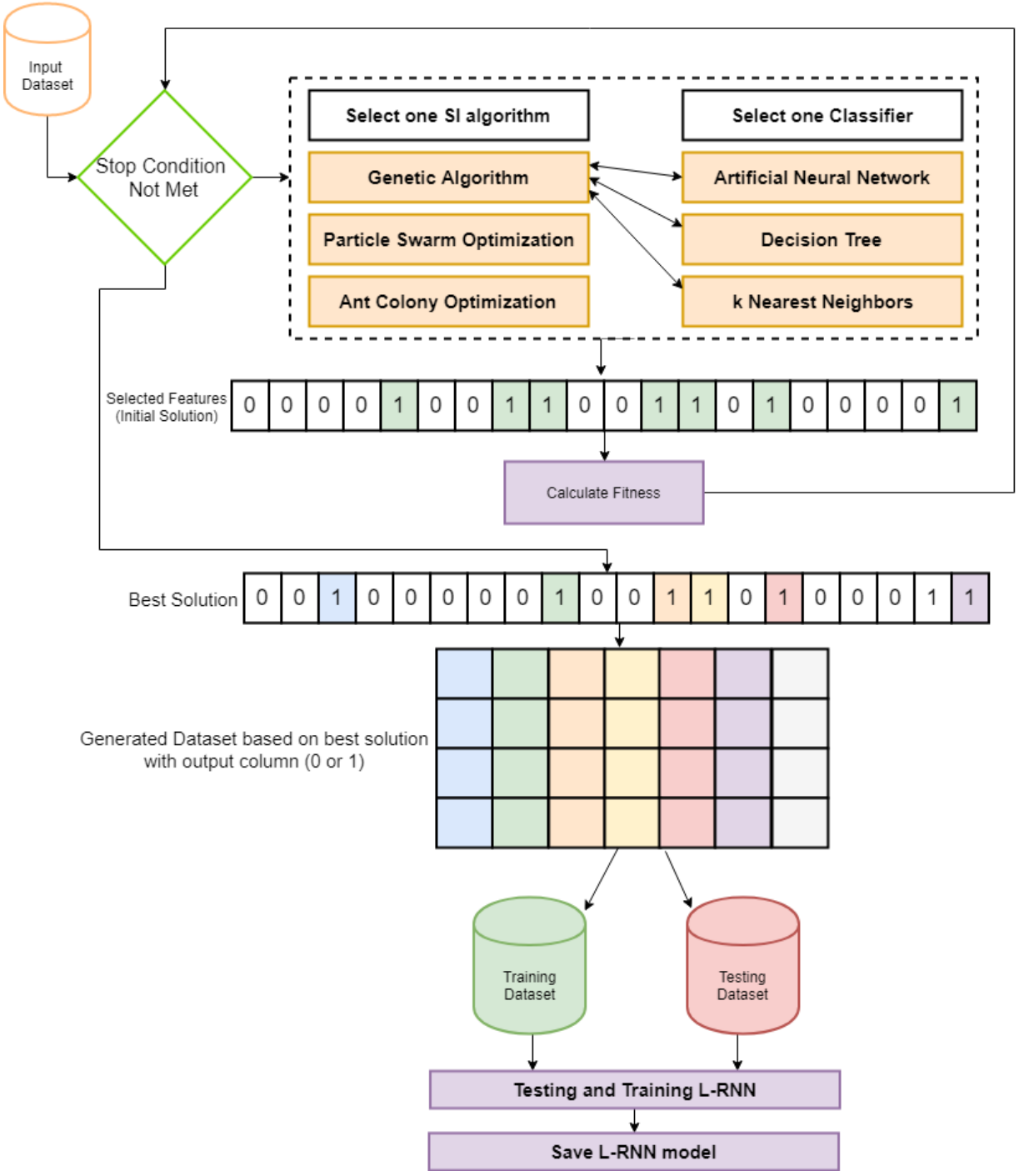

Figure 4.1: Software Fault Prediction model

Figure 4.2 explains how the majority FS algorithm works, below the explanation:

1. Choose set of datasets to be the input for SFP model.
2. Select SI algorithm (PSO, ACO, GA)
3. Send the selected features to evaluation phase.
4. Select classification algorithm (kNN, DT, ANN)
5. Keep iterating the model with same selection for x number of iterations to give the chance classifier component to learn the data.
6. Save features subset, fitness value, classification algorithm used and SI algorithm then try another classifier and SI combination, as shown in Figure 4.2 the result of this phase will be a solution with only specific number of features selected, with the fitness value of this solution, this reduced solution shall improve the accuracy and performance of deep learning component which in result will improve the overall SFP accuracy and performance.
7. Calculate how many times each feature got selected in the 9 solutions, the hypothesis here that the most selected feature in the solution is the most critical feature that will drive the system accuracy.
8. Rank each feature. Ranking the feature calculated in a simple way by dividing the sum of feature occurrences in all solutions on the number of solutions.
9. Generate new feature subset using majority voting. In this step the x highest ranked features has been taken to generate the best solution and passed it to data reduction phase, the x number here is questionable, what value should it take, two ways have been tested as follows:

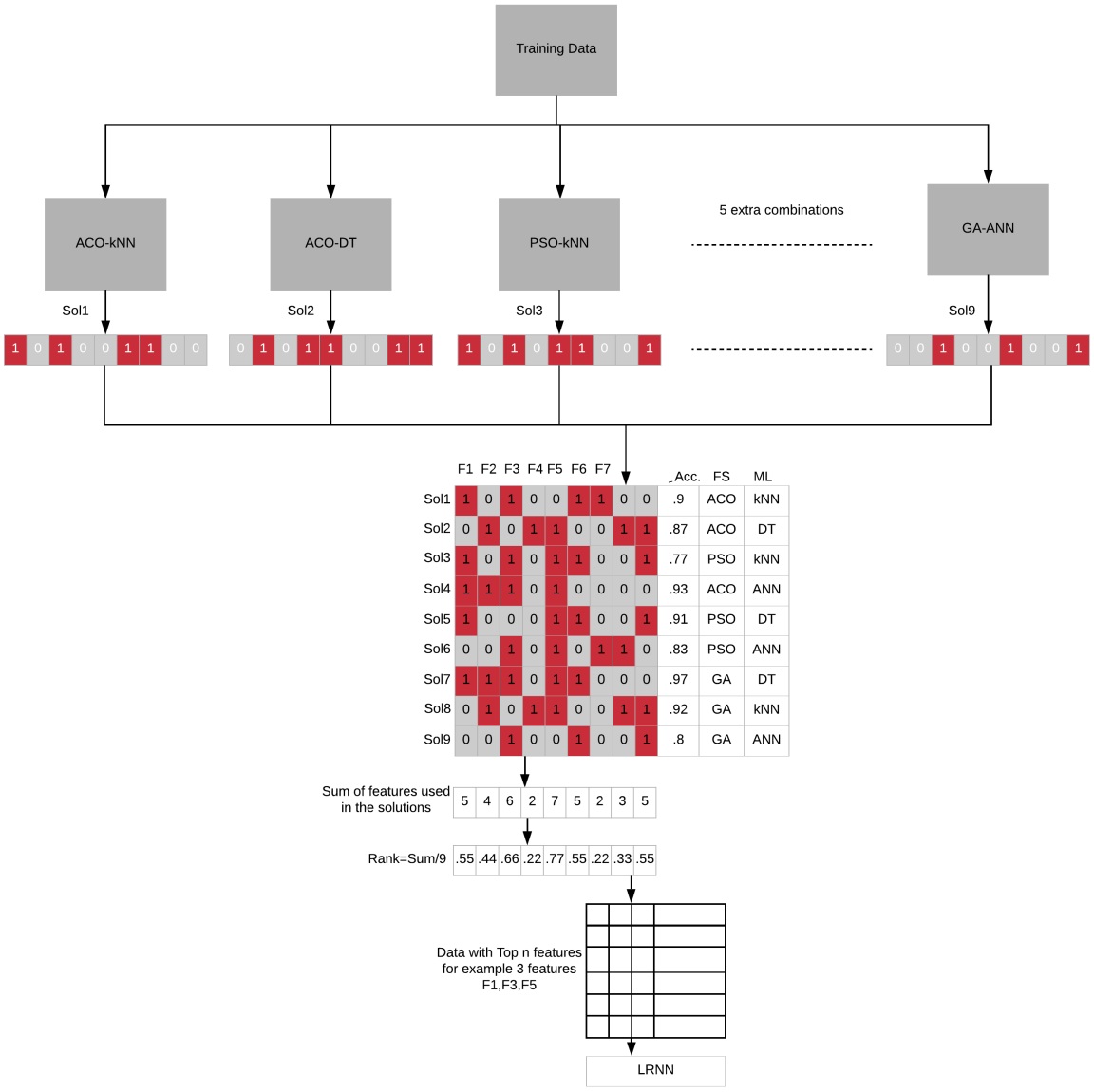

Figure 4.2: Majority voting Software Fault Prediction model

- Best solution number of features, x will be equal to the number of selected features of the best solution among the 9 solutions, in our example solution 7 has the best accuracy .97 with 5 features selected, so x will be 5.
- The average number of selected features among all the solutions, in our example the solutions selected features numbers are (5,5,5,4,4,4,5,5,3) respectively, the average number of selected features is $40/9 = 4.44$ so x will be 4.

10. Data reduction phase, in this phase the dataset got filtered based on the selected features along with the output class and passed it to deep learning phase.
11. Deep learning phase, the reduced data has been passed to layered recurrent neural network (L-RNN) component which is used as a classification technique for SFP problem.

4.1.3 Analysis phase

In this phase we compared the 10 different SFP models based on AUC metric as follows:

- Calculate the AUC (area under curve) of SFP model resulted from training the model on the data generated from the majority voting phase.
- Calculate the AUC of SFP model resulted from training the model on the data generated from each combination alone, for example solution 1 was generated using ACO-kNN, the data will be reduced using this solution then passed to deep learning phase, accuracy will be calculated.

- Sort the 10 SFP models based on avg. AUC (9 generated from each solution alone and another one from the majority voting solution) and check which combination resulted in the best AUC.

This research can enhance the performance of ML recommendation frameworks by applying feature selection algorithms to reduce number of meta-features. The results of the proposed approach above can be used in other dimension, instead of only enhancing the SFP model, we can start recommending the best FS combination to the project based on its meta features, as Dores, Alves and Ruiz [22] proposed a framework for recommending the best ML of certain software project, the proposed model as follows:

1. Choose set of datasets to be the input for recommendation system.
2. Define Meta-Features to be used and extract them from each dataset entered
3. Define input algorithms, define the set of algorithms that the system will run on every dataset entered and the performance metrics and results will be saved.
4. Define the performance metric to evaluate each algorithm metric.
5. Building the Meta-Database which embodies the knowledge of the relationship between meta-features and algorithmic predictive performance.
6. Choosing meta-learner to build recommendation model.
7. All of the above steps are to build the recommendation model itself, at last enter new dataset, extract its meta-features, and a result the best algorithm to be recommended.

As seen in the above model [22] all meta-features of each dataset was extracted. At least 50 datasets need to be used as recommended for significant meta-learning

analysis [22]. Each dataset contains 20 meta-features. Which results in huge number of features to be measured and analyzed. This may result in performance issue which contradicts with the whole purpose of fault prediction models that aims to spare and optimize QA team time.

The aim of this thesis is to apply feature selection algorithms to select the dominant meta-features. Then try to find the best features that can be used in recommendation model for SFP.

Chapter 5

Results

In this chapter, the details of the experimental results were reported and discussed; the results of the nine SFP models were analyzed, followed by the results the proposed approach where the the majority voting technique was used to select the feature subset. In this thesis, nineteen different datasets from the PROMISE repository were used.

The remainder of this chapter is organized as follows: Section 5.1 describes the experimental setup. Section 5.2 provides a brief overview of the used datasets for evaluation and comparison purposes, followed by the details of the used performance metrics in Section 5.3. Finally, in Section 5.4, the evaluation results are reported and deeply analyzed and discussed.

5.1 Experimental setup

In this thesis, all experiments were run on a machine with Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz X 8 processors and 32GB RAM, and all algorithms were implemented using Matlab. The details of the parameter settings of the used algorithms are listed in the below tables. All parameters were set based on

preliminary runs.

The used parameters for PSO explained in table 5.1.

Table 5.1: PSO Parameters

Parameters	Values
Number of iterations	3000
Degree of influence (c1&c2)	1.5
vmax	1.0
vmin	0
Inertia Weight (w)	0.8

The used parameters in ACO algorithms were explained in Table 5.2.

Table 5.2: ACO Parameters

Parameters	Values
Number of iterations	3000
Population size (number of ants)	20
Initial pheromone	1.0
Pheromone Exponential Weight (α)	0.8
Heuristic Exponential Weight (β)	0.8
Evaporation Rate	0.6

Table 5.3 presented the basic parameters that were used for GA algorithm.

Table 5.3: GA Parameters

Parameters	Values
Number of iterations	3000
Population size	40
Crossover rate	0.7
Mutation rate	0.1
Selection type	Roulette Wheel Selection
Crossover type	single, double, or uniform

The L-RNN parameters and configurations were explained in Table 5.4.

Table 5.4: L-RNN Parameters

Parameters	Values
Number of iterations	1000
Number neurons in Input layer	Number of input data
Number neurons in Hidden layer	$\frac{Number\ of\ input\ data}{2}$
Number neurons in Output layer	1.0
Threshold value to transfer output	0.5

The experiments has been run as follows:

- For each SI-Classifier combination, keep iterating for a predefined number of iterations to obtain the optimized solution. Each solution is represented as a binary vector of 20 feature as shown in Figure 5.1, where the value 1 means that this feature is selected, and 0 means that the feature is not selected.
- Each solution is evaluated based on a fitness function that combines both classifier error rate and the selection ratio as shown in Equation 5.1.

$$Fitness = E * (1 + (\beta * \frac{|SF|}{|NF|})) \tag{5.1}$$

where E is the error rate as show in equation 5.2, β is predefined variable for this work is 0.5 based on literature, SF is the number of selected features and NF is the total number of features.

$$E = \frac{FP + FN}{TP + TN + FP + FN} \tag{5.2}$$

It is worth mentioning here that the fitness value take two factors into consideration. First, the percentage of selected features, FS algorithms aims to reduce the number of the selected features to enhance the performance of the classification process later on, second the classification accuracy that is expressed in terms of the error factor (E).

- The whole FS process has been repeated for twenty times for each combination on each dataset, to reduce the probability that an algorithm started with a bad solution and trying to optimize a bad solution, in SI algorithm the initial solution is important, generally SI algorithms try to solve the problem of getting stuck in local optima, however, the initial solution that the algorithms starts with may affect the algorithm accuracy.

- L-RNN classifier has been run for 100 times for each SI-Classifier combination and the average AUC of the all runs has been calculated.

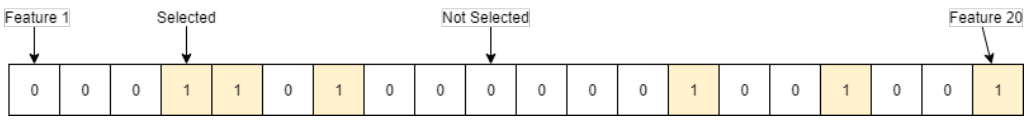

Figure 5.1: Example of features vector.

5.2 Datasets

Many SFP datasets are publicly available such as PROMISE [65], NASA [68] and AEEEM [21]. Dataset is a matrix, each row represents a software module/class, each column represents a software metric such as CK [19, 20] object oriented metrics. Each matrix entity represents a normalized value of the software metric for this software module/class. The last column is the output a column, which is a binary value that states if this module/class is faulty or not.

In this work, nineteen well known datasets from PROMISE repository were used for testing and evaluation purposes. The PROMOSE datasets is one of the most frequently used data sets in the literature [53]. PROMISE datasets are clean and noise free. Datasets varies in size (i.e., 109 to 909 instances) and varies in the percentage of the defective instances (i.e., 2.2% to 98.8%). Each dataset contains a 20 object oriented metric that represent the features for the SFP model, those metrics are (wmc, dit, noc, cbo, rfc, lcom, ca, ce, npm, lcom3, loc, dam, moa, mfa, cam, ic, cbm, amc, max_cc, avg_cc), those metrics are explained in details in section 2.2, table 5.5 describes the details of PROMISE datasets.

Industrial datasets were not used since they are not publicly available, which make it hard for other researchers to reproduce and reuse the results built on top of them. It was observed that very few studies used them for evaluating the effectiveness of the ML techniques [53].

Table 5.5: PROMISE datasets details

Dataset	# of Instances	# of faulty instances	Rate of faulty instances
ant-1.7	745	166	0.223
camel-1.0	339	13	0.038
camel-1.2	608	216	0.355
camel-1.4	872	145	0.166
camel-1.6	965	188	0.195
jedit-3.4	272	90	0.331
jedit-4.0	306	75	0.245
jedit-4.2	367	48	0.131
jedit-4.3	492	11	0.022
log4j-1.0	135	34	0.252
log4j-1.1	109	37	0.339
log4j-1.2	205	189	0.922
lucene-2.0	195	91	0.467
lucene-2.2	247	144	0.583
lucene-2.4	340	203	0.597
xalan-2.4	723	110	0.152
xalan-2.5	803	387	0.482
xalan-2.6	885	411	0.464
xalan-2.7	909	898	0.988

5.3 Performance Metrics

A set of performance metrics available for use to evaluate the performance of the proposed model such as classification accuracy, area under curve (AUC), F-measure, precision and recall. Many studies [81, 47] suggest to use AUC for evaluating classifiers, since AUC is not affected in case of changing data distributions, so AUC metric will be used in this work for comparing and evaluating the results of the different models.

AUC depends on a trade-off between True Positive rate against False Positive rate. It can be calculated out of confusion matrix in figure 5.6 using two values, Specificity (TN rate) and Sensitivity (TP rate) as follows:

$$Specificity = \frac{TN}{N} \tag{5.3}$$

$$Sensitivity = \frac{TP}{P} \tag{5.4}$$

The usage of AUC metric can be justified that in SFP domain classifying a non-faulty module as faulty will result in extra testing for this module. However, classifying a faulty module as faulty may cause a big damage to the software product, a potential bug should have been reported, fixed and tested were ignored.

Moreover, datasets are imbalanced, for some datasets, 95% of the instances are non-faulty and only 5% are faulty, so classifying all the non-faulty modules correctly and all the faulty modules incorrectly will result in 95% accuracy, which is good accuracy, however very bad in terms of efficiency. That's why AUC is a better performance metric.

Figure 5.2 shows the proposed rules to evaluate any classifier using AUC [38]. AUC measures enable researchers to generalize the results even if the data distribution is changed [46].

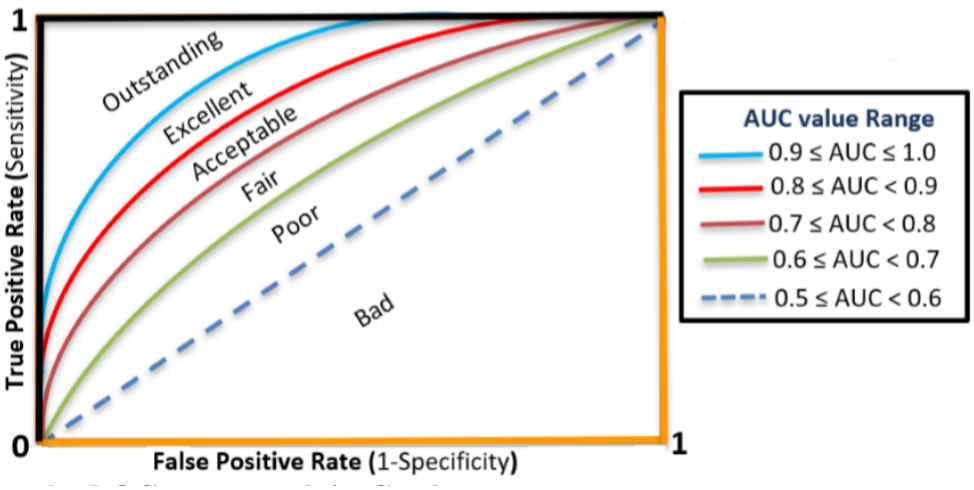

Figure 5.2: ROC curves and AUC values

		Predicted Class	
		Class A	Not Class A
Actual Class	Class A	TP	FN
	Not Class A	FP	TN

Table 5.6: Confusion Matrix for binary class problem

5.4 Experimental Results

In this section a deep analysis for of the obtained results is presented. The comparisons are presented in three phases; the results obtained from running the L-RNN classifier on all the datasets without feature selection are recorded in the first phase, then the results of running the L-RNN using the reduced datasets after applying FS combination on those datasets are reported, it also describes the results of running the majority voting algorithm on every dataset.

Finally, features importance was analyzed by comparing the frequency of selecting each feature using the ten algorithms, assuming that the importance of the feature is expressed according to its frequency in the final feature subsets.

5.4.1 Results without FS

Table 5.7 shows the average AUC of training the L-RNN model using full datasets (without applying FS) on all the 19 datasets. The results varies from excellent to fair on different datasets (based on the classification presented in Figure 5.2). For example, the model achieved 100% AUC for the log4j_1_1 dataset, in contrast 0.72 AUC for the xalan_2_6 dataset, the average AUC on all datasets is 0.84 and in 4th place compared to other FS combinations.

The main drawback of using full datasets in FS models is that the runtime of training the model without reducing the dimensionality is high. Intuitively, the assumption would be that running FS algorithm plus running the classifier would consume less time in comparison with running the classifier alone. However, this assumption is proven to be incorrect, as the SFP model is supposed to be run while developing each software feature or bug solving multiple times, so the cumulative runtime of the SFP model with FS will definitely be less than runtime the SFP model without FS.

Table 5.7: Avg. AUC results of L-RNN on all datasets without FS

Dataset	AUC
ANT	0.95
Camel_1_0	0.80
Camel_1_2	0.74
Camel_1_4	0.97
Camel_1_6	0.76
Jedit_3_4	0.83
Jedit_4_0	0.80
Jedit_4_2	0.75
Jedit_4_3	0.77
log4j_1_0	0.96
log4j_1_1	1.00
log4j_1_2	0.97
lucene_2_0	0.85
lucene_2_2	0.79
lucene_2_4	0.95
xalan_2_4	0.78
xalan_2_5	0.75
xalan_2_6	0.72
xalan_2_7	0.74

5.4.2 Results with FS

Table 5.9 shows the results of executing each FS combination on every dataset. Obviously, it can be observed that the results varies for each combination which

emphasize on the importance of choosing the best fit FS algorithm.

For PSO, ANN obtained the highest results with average 0.83, a deeper look into the results per dataset shows that for many of the datasets this combination can provide us with a near optimal solution to give near 100% AUC, for DT the results are relatively low, kNN comes in the second place when running with PSO with average 0.81.

For GA, ANN obtain the highest results with average 0.85, the GA-ANN combination is the best combination among all the nine combinations, the results for 13 datasets out of 19 is higher than or equal to 0.85, which are very good results, only one outlier for GA-ANN for the xalan_2_4 dataset with a very low average AUC equals to 0.35, for DT is worst combination among all the nine combinations with average AUC equals to 0.64, for kNN the average AUC is 0.79. For ACO, the best classifier to work with ACO is kNN with 0.84 average AUC, this combination comes at the second stage among all the nine combinations, in general ACO works well with all three classifiers with 0.81 and 0.80 average AUC for ANN and DT respectively.

Table 5.8 shows that ANN is the best classifier, which works well with all the SI search algorithms, and best to work with GA algorithm, for kNN it comes in the second place to work well also with all the SI search algorithms. However, kNN works best with ACO, for DT comes in third and last place as the worst classifier, it works worst with GA algorithm.

Table 5.8: Results grouped by classifier

Classifier	AVG. AUC
ANN	0.83
DT	0.74
kNN	0.81

Based on the above results we recommend the following:

Table 5.9: Results with FS

Algorithm / Dataset	PSO-ANN	PSO-DT	PSO-kNN	GA-ANN	GA-DT	GA-kNN	ACO-ANN	ACO-DT	ACO-kNN
ANT	0.72	0.99	1.00	0.87	0.50	0.50	1.00	0.98	0.90
Camel_1_0	0.75	0.71	0.77	0.79	0.73	0.50	0.89	0.90	0.83
Camel_1_2	0.89	0.65	0.98	0.64	0.50	0.88	0.91	1.00	1.00
Camel_1_4	0.92	1.00	0.86	0.83	0.50	0.68	0.86	0.84	0.86
Camel_1_6	0.98	0.99	0.91	0.92	0.50	0.58	0.85	0.83	0.75
Jedit_3_4	0.73	0.79	0.94	0.90	0.94	0.96	0.97	0.79	1.00
Jedit_4_0	0.66	0.77	0.68	0.91	0.84	1.00	0.68	0.62	0.77
Jedit_4_2	1.00	0.70	0.84	0.92	0.80	0.78	0.39	0.72	0.98
Jedit_4_3	0.81	0.70	0.77	0.85	0.84	0.68	0.96	0.82	0.62
log4j_1_0	0.97	0.67	0.75	0.97	0.61	1.00	0.55	0.94	1.00
log4j_1_1	0.77	0.97	0.49	1.00	0.80	0.99	0.84	1.00	0.95
log4j_1_2	0.84	0.54	0.99	0.96	0.64	0.86	0.72	0.89	0.75
lucene_2_0	0.83	1.00	0.83	0.98	0.53	0.89	0.97	0.53	0.78
lucene_2_2	0.99	0.04	0.52	0.92	0.64	0.73	0.67	0.05	0.96
lucene_2_4	0.89	0.88	0.67	0.94	0.60	0.66	0.70	0.94	0.74
xalan_2_4	0.75	0.89	0.89	0.35	0.53	0.92	0.84	0.84	0.29
xalan_2_5	0.86	0.74	0.88	0.79	0.58	0.82	0.85	0.96	0.80
xalan_2_6	0.84	0.75	0.77	0.85	0.50	0.67	0.86	0.67	0.89
xalan_2_7	0.56	0.87	0.77	0.80	0.50	0.87	0.94	0.85	0.99
Average	0.83	0.77	0.81	0.85	0.64	0.79	0.81	0.80	0.84
Average per SI Algo.		0.80			0.76			0.81	

- The best combination is GA-ANN.
- The best SI search algorithm to work with all three classifiers is ACO.
- The best ML algorithm to work with all three SI search algorithms is ANN.
- The worst combination is GA-DT.

5.4.3 Results of majority algorithm

Majority algorithm is a brand new approach FS algorithm proposed in this thesis research, which considers all other FS combinations when building its solution, the basic idea is to take the highest used features in the other FS algorithms solutions. The number of features to be taken is equal to the average number of selected features of all other combinations.

Based on the experiments majority algorithm proved to be very stable and robust algorithm, which performs well on all the datasets, table 5.10 shows the results of running the majority algorithm in all the 19 datasets. The average AUC for running majority algorithm is 0.95 which outperforms the best algorithm GA-ANN, the algorithm obtained 100% average AUC 6 datasets of 19, now we will describe the rank of majority algorithm in each dataset.

- ANT: majority comes in the 6th place with average AUC 0.88 and 5 selected features with average selected features 4.78.
- Camel_1_0: majority comes in the 1st place with average AUC 0.92 and 5 selected features with average selected features 4.34.
- Camel_1_2: majority comes in the 3rd place with average AUC 0.98 and 5 selected features with average selected features 4.89.
- Camel_1_4: majority comes in the 2nd place with average AUC 0.99 and 5 selected features with average selected features 4.45.
- Camel_1_6: majority comes in the 3rd place with average AUC 0.94 and 6 selected features with average selected features 5.11.
- Jedit_3_4: majority comes in the 1st place with average AUC 1.00 and 7 selected features with average selected features 6.78.
- Jedit_4_0: majority comes in the 1st place with average AUC 1.00 and 5 selected features with average selected features 4.56.
- Jedit_4_2: majority comes in the 1st place with average AUC 1.00 and 6 selected features with average selected features 5.33.
- Jedit_4_3: majority comes in the 2nd place with average AUC 0.90 and 5 selected features with average selected features 4.78.

- log4j_1_0: majority comes in the 1st place with average AUC 1.00 and 6 selected features with average selected features 5.33.
- log4j_1_1: majority comes in the 4th place with average AUC 0.96 and 6 selected features with average selected features 5.22.
- log4j_1_2: majority comes in the 4th place with average AUC 0.86 and 7 selected features with average selected features 6.22.
- lucene_2_0: majority comes in the 1st place with average AUC 1.00 and 6 selected features with average selected features 5.33.
- lucene_2_2: majority comes in the 2nd place with average AUC 0.97 and 5 selected features with average selected features 4.78.
- lucene_2_4: majority comes in the 3rd place with average AUC 0.90 and 7 selected features with average selected features 6.56.
- xalan_2_4: majority comes in the 1st place with average AUC 0.93 and 5 selected features with average selected features 4.78.
- xalan_2_5: majority comes in the 2nd place with average AUC 0.89 and 5 selected features with average selected features 4.33.
- xalan_2_6: majority comes in the 1st place with average AUC 0.96 and 5 selected features with average selected features 5.00.
- xalan_2_7: majority comes in the 1st place with average AUC 1.00 and 5 selected features with average selected features 4.889.

The ranking shows that the majority algorithm came in the 1st place for 9 datasets out of 19, and 4 times in the 2nd place, which emphasize that the majority voting approach outperforms other FS approaches.

Table 5.10: Results majority

Dataset	AUC
ANT	0.88
Camel_1_0	0.92
Camel_1_2	0.98
Camel_1_4	0.99
Camel_1_6	0.94
Jedit_3_4	1.00
Jedit_4_0	1.00
Jedit_4_2	1.00
Jedit_4_3	0.90
log4j_1_0	1.00
log4j_1_1	0.96
log4j_1_2	0.86
lucene_2_0	1.00
lucene_2_2	0.97
lucene_2_4	0.90
xalan_2_4	0.93
xalan_2_5	0.89
xalan_2_6	0.96
xalan_2_7	1.00
Average	0.95

Statistical comparison using t-test with a significance level of 0.05 was conducted as to compare majorty model with other FS models was done independently against all 10 FS models. Table 5.11 presents the p-values of the obtained results and mean difference. In this table, a p-value less than 0.05 indicates that there

is a statistical difference between results. The average increase when comparing majority model with all 10 other models was found to be statistically significant across all models.

Table 5.11: P-value results based on t-test

	Majority	
Model	μ_1 - μ_2	P-value
Without FS	0.12	0.00042
PSO-ANN	0.12	0.00058
PSO-DT	0.18	0.00286
PSO-kNN	0.15	0.00099
GA-ANN	0.10	0.01127
GA-DT	0.32	1.4E-08
GA-kNN	0.16	0.00011
ACO-ANN	0.14	0.00261
ACO-DT	0.15	0.01305
ACO-kNN	0.12	0.00454

Table 5.12 compares results of majority voting FS model with several methods in the literature. Turabieh [76] used random function to select from three FS methods coming from combining ANN with three search algorithms (GA, PSO and ACO), L-RNN was used as evaluator. Erturk & Sezer [28] used only 3 features based on the recommendation from literature [62], namely (cbo, wmc and rfc) with L-RNN as classifier. The comparison shows that majority voting technique came in first place 18 times out of 19 with 0.11, 0.17, 0.14 increase in average from results obtained by Turabieh, Erturk & Sezer respectively.

Table 5.12: Comparison between majority voting technique and the sate-of-the art methods based on the average AUC values

	Our Results	Turabieh	(Erturk and Sezer, 2016)	
Dataset	Majority Voting	Turabieh	ANN	ANFIS
ANT	0.88	0.88	0.85	0.82
Camel_1_0	0.92	0.91	0.92	0.89
Camel_1_2	0.98	0.65	0.60	0.60
Camel_1_4	0.99	0.78	0.79	0.81
Camel_1_6	0.94	0.69	0.68	0.71
Jedit_3_4	1	0.92	0.88	0.90
Jedit_4_0	1	0.87	0.82	0.78
Jedit_4_2	1	0.89	0.88	0.98
Jedit_4_3	0.9	0.91	0.46	0.91
log4j_1_0	1	0.88	0.89	0.89
log4j_1_1	0.96	0.91	0.90	0.90
log4j_1_2	0.86	0.85	0.78	0.77
lucene_2_0	1	0.87	0.85	0.87
lucene_2_2	0.97	0.82	0.76	0.75
lucene_2_4	0.9	0.83	0.82	0.81
xalan_2_4	0.93	0.83	0.82	0.82
xalan_2_5	0.89	0.75	0.67	0.66
xalan_2_6	0.96	0.71	0.68	0.68
xalan_2_7	1	0.93	0.82	0.86
Average	0.95	0.84	0.78	0.81

Table 5.14 shows the number of selected features per dataset per algorithm, the results show that in general ACO algorithm tends to select most features, on

Table 5.13: Example of features histogram to generate majority solution

Feature number	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
Feature count	8	7	6	5	5	5	3	2	2	1

average for ACO per classifier more than 6 features has been selected for each solution, in addition to GA_ANN with the average equals to 8 selected feature which is the highest average, the more the features are selected the higher the cost of running the algorithm.

Challenges. While designing an SFP model that depends on a Majority FS algorithm it is important to consider two main issues. First the time needed to run all other combinations before preparing the majority solution. It is important to always think of the FS step in any SFP model as preliminary stage, that selects the most important features of this project, then the SFP model will be using those solutions many many times on the later stages while developing the software project.

The second issue is in case of having two features with same frequency then which feature to take, for example, table 5.13 shows 10 features and how many each feature is selected in the 9 other combinations, assuming that the majority solution need to consist of 4 features, then we will choose F1, F2, F3 and on of these F4, F5, or F6, so which one to choose. In this research all the combinations were tried. the best combination was taken, the results varies a lot between each solution which emphasize on the fact that some features work better with other features.

Table 5.14: Number of features selected per Algorithm per dataset

Dataset/Algo.	PSO-ANN	PSO-DT	PSO-kNN	GA-ANN	GA-DT	GA-kNN	ACO-ANN	ACO-DT	ACO-kNN	Majority
ANT	4	4	4	6	1	1	7	10	6	5
Camel_1_0	4	4	4	6	3	1	6	5	6	5
Camel_1_2	4	4	4	9	1	3	6	8	5	5
Camel_1_4	4	4	4	6	1	3	5	7	6	5
Camel_1_6	4	4	4	10	1	2	10	5	6	6
Jedit_3_4	4	4	4	9	3	5	9	6	8	7
Jedit_4_0	4	4	4	8	2	3	6	5	5	5
Jedit_4_2	4	4	4	11	2	3	8	5	7	6
Jedit_4_3	4	4	4	5	2	3	10	6	5	5
log4j_1_0	4	4	4	7	2	5	11	6	5	6
log4j_1_1	4	4	4	8	2	2	9	7	7	6
log4j_1_2	4	4	4	11	2	3	10	11	7	7
lucene_2_0	4	4	4	10	2	4	6	7	7	6
lucene_2_2	4	4	4	8	2	3	8	5	5	5
lucene_2_4	4	4	4	10	2	4	11	10	10	7
xalan_2_4	4	4	4	6	2	3	7	8	5	5
xalan_2_5	4	4	4	3	2	3	7	5	7	5
xalan_2_6	4	4	4	8	1	4	6	8	6	5
xalan_2_7	4	4	4	11	1	3	7	5	5	6
Average	4	4	4	8	1.79	3.05	7.84	6.79	6.21	5.63

Table 5.15 shows the solutions generated from each FS combination, those solutions varies from run to run, however, those solution can be used as a reference for this thesis results.

Table 5.15: Selected solution for each algorithm per dataset

Dataset \ Algorithm	PSO-ANN	PSO-DT	PSO-kNN	GA-ANN	GA-DT	GA-kNN	ACO-ANN	ACO-DT	ACO-kNN	Majority
ANT	[11,9,8,2]	[17,2,13,16]	[1,15,20,11]	[5,11,13,14,15,16]	[19]	[1;5]	[16,7,17,15,19,10,11]	[3,17,9,18,12,16,2,8,5,20]	[11,12,16,19,8,9]	[11,16,2,8,9]
Camel_1_0	[1,3,16,13]	[19,14,6,13]	[8,9,19,16]	[4,7,9,10,15,16]	[9,12,17]	[10]	[17,2,10,18,12,4]	[6,14,9,11,19]	[10,6,3,14,11,13]	[9,10,19,16,14]
Camel_1_2	[11,14,3,20]	[16,2,17,12]	[17,7,9,4]	[1,3,4,6,9,11,12,13,17]	[13]	[7,8,9]	[8,20,9,5,14,16]	[16,20,2,17,9,19,15,11]	[2,15,8,9,14]	[9,17,2,8,11]
Camel_1_4	[11,14,19,9]	[13,17,2,16]	[13,16,9,8]	[1,2,4,8,14,19]	[16]	[5,15,16]	[15,16,19,4,14]	[9,10,13,14,1,7,2]	[1,10,16,14,19,13]	[16,14,13,19,1]
Camel_1_6	[8,4,9,1]	[12,19,16,17]	[2,20,18,4]	[1,4,6,7,8,10,11,13,14,16]	[2]	[6,7]	[1,18,12,16,7,14,20,13,10,19]	[19,12,9,16,13]	[12,20,4,18,6,7]	[4,7,12,16,18,0]
Jedit_3_4	[11,2,1,12]	[7,12,16,4]	[17,2,1,18]	[1,4,6,10,12,14,15,17,19]	[15,17,19]	[1,2,12,17,18]	[9,18,15,17,11,5,1,3,19]	[9,20,7,18,17,5]	[12,6,2,1,17,19,8,14]	[17,1,12,2,18,19,15]
Jedit_4_0	[2,6,8,12]	[14,2,1,12]	[2,16,10,9]	[6,8,9,10,12,14,17,20]	[1,12]	[17,19,20]	[2,19,4,13,16,9]	[18,12,13,3,17]	[17,14,3,9,20]	[12,2,9,17,14]
Jedit_4_2	[7,5,15,1]	[17,16,7,13]	[7,16,13,20]	[3,4,7,8,9,10,11,14,15,16,17]	[1,12]	[4,13,20]	[3,5,10,4,12,17,7,15]	[2,19,13,5,9]	[20,10,17,2,9,18,1]	[7,13,17,20,1,5]
Jedit_4_3	[8,17,13,12]	[2,12,13,9]	[7,1,10,13]	[2,7,8,12,17]	[1,12]	[4,13,20]	[10,12,17,15,4,19,14,9,20,2]	[12,2,13,20,9,10]	[8,16,18,5,3]	[12,13,2,20,9]
log4j_1_0	[7,9,16,5]	[3,7,14,9]	[9,4,5,3]	[1,4,5,9,14,16,18]	[7,9]	[1,4,5,9,19]	[9,6,13,1,7,5,12,18,16,8,2]	[7,16,19,13,1,18]	[10,20,19,5,9]	[9,5,7,1,16,4]
log4j_1_1	[6,9,5,13]	[6,1,14,16]	[9,5,14,3]	[1,3,5,9,11,13,18,20]	[7,9]	[5,9]	[16,7,10,1,5,12,11,18,6]	[3,7,16,17,1,8,5]	[20,1,16,7,5,9,14]	[5,9,1,7,16,14]
log4j_1_2	[12,2,17,1]	[16,3,17,13]	[7,2,1,5]	[1,3,4,6,9,10,11,12,14,16,17]	[7,9]	[6,8,15]	[6,9,7,4,13,16,19,10,17,8]	[19,20,2,15,3,4,18,14,7,6,17]	[5,17,2,4,10,16,13]	[17,2,4,6,7,16,9]
lucene_2_0	[19,18,17,1]	[8,17,14,16]	[19,18,14,5]	[4,5,6,7,10,11,14,15,16,17]	[7,9]	[3,5,17,18]	[15,8,5,2,16,20]	[20,17,13,8,5,3,10]	[13,18,1,14,5,8,6]	[5,17,8,14,18,16]
lucene_2_2	[2,9,1,5]	[9,5,13,15]	[18,1,15,10]	[1,3,6,9,12,18,19,20]	[7,9]	[4,9,16]	[5,9,8,1,11,10,15,6]	[12,1,14,18,2]	[4,12,1,16,17]	[1,9,5,12,15]
lucene_2_4	[5,7,16,4]	[2,13,8,16]	[5,15,4,3]	[2,4,5,6,10,12,16,17,18,20]	[7,18]	[3,4,5,16]	[18,1,10,7,13,4,14,16,8,5,19]	[9,14,10,3,13,7,11,8,2,18]	[2,19,9,15,7,5,8,20,4,6]	[4,5,7,16,2,8,18]
xalan_2_4	[4,11,2,7]	[17,3,16,2]	[7,2,4,11]	[2,7,8,11,12,15]	[7,9]	[7,8,11]	[5,2,11,20,10,3,13]	[9,18,20,16,14,1,17,8]	[11,16,8,12,15]	[11,2,7,8,16]
xalan_2_5	[14,19,5,4]	[10,17,7,2]	[15,14,10,20]	[5,8,14]	[7,9]	[8,9,18]	[17,20,11,14,10,9,4]	[18,11,14,8,16]	[20,8,1,17,4,10,9]	[14,8,9,10,17]
xalan_2_6	[11,5,8,17]	[17,10,14,7]	[7,10,17,4]	[2,3,5,8,9,13,15,18]	[14]	[1,4,10,17]	[3,18,9,17,11,4]	[16,14,4,15,9,13,18,3]	[10,15,8,1,17,14]	[17,4,10,14,18]
xalan_2_7	[18,1,9,2]	[5,17,4,18]	[5,18,20,8]	[1,2,3,5,6,7,9,11,13,14,15]	[14]	[5,8,18]	[6,10,2,8,17,18,1]	[8,12,14,11,19]	[6,18,12,5,11]	[18,5,8,14,2,6]

5.4.4 Most Used Features

Figure 5.3 shows the count of how many each feature is selected in the 9 different combinations solutions, below the analysis:

- npm (number of public methods) and got selected 69 times.
- cbm (coupling between methods) which is object oriented metric and got selected 57 times.
- ic (inheritance coupling) which is object oriented metric and got selected 54 times.
- wmc (weighted method per class) which is object oriented metric and got selected 50 times.
- rfc (response for a class) which is object oriented metric and got selected 50 times.
- ca (afferent couplings) which is object oriented metric and got selected 50 times.

From the above results it can be observed that the top selected 6 metrics are object oriented metrics, based on our assumption the most used features are the most important features, and all of them are OO metrics as mentioned in other references like [20, 19, 62]

Table 5.16 shows the histogram of features selection per dataset, this table describes the importance of each feature to this dataset, for example feature npm selected from 8 algorithms from our 9 algorithms for dataset log4j_1_0 and feature rfc selected from 7 algorithms from our 9 algorithms.

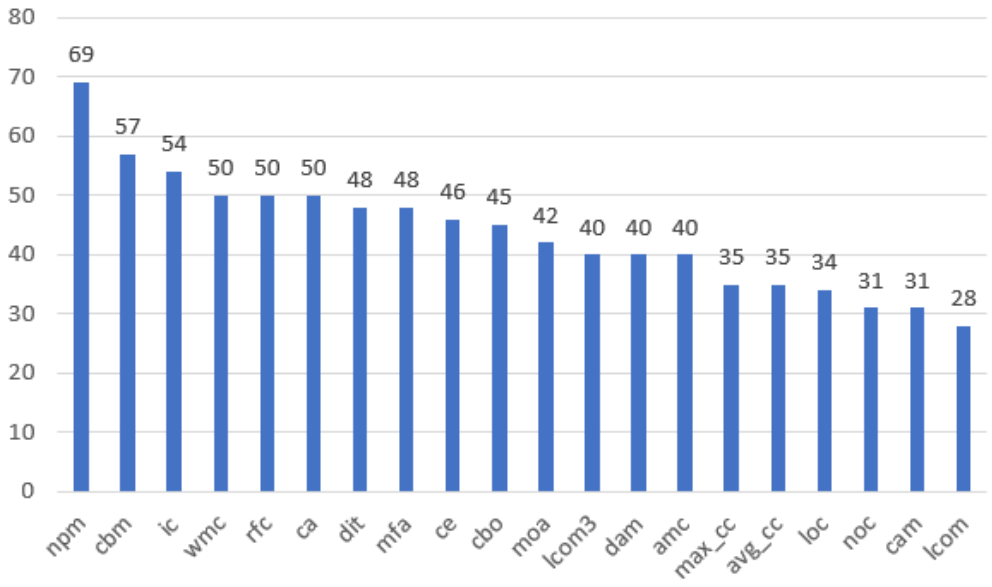

Figure 5.3: Features selection histogram.

5.4.5 Summary

- SFP model without FS achieved 0.84 average AUC on all datasets.
- The best FS combination is GA-ANN with average AUC 0.85.
- The best search algorithm to work with all three classifiers is ACO.
- The best classifier to work with all three search algorithms is ANN.
- The worst FS combination is GA-DT.
- Majority voting technique outperforms all other techniques with minimum 0.10 increase in average AUC with average AUC equals to 0.95.
- ACO search algorithm tends to choose more features than PSO and GA with average selected 7 features compared to average 4 features and 4.28 features for PSO and GA respectively.

Table 5.16: Features histogram

Feature / Dataset	wmc	dit	noc	cbo	rfc	lcom	ca	ce	npm	lcom3	loc	dam	moa	mfa	cam	ic	cbm	amc	max_cc	avg_cc
ANT	2	3	1	0	2	0	1	3	3	1	5	2	2	1	3	5	3	1	3	2
Camel_1_0	0	1	2	3	0	3	2	1	4	4	2	2	3	3	1	2	2	1	3	0
Camel_1_2	1	3	2	2	1	1	2	3	6	0	3	2	2	3	2	3	4	0	1	3
Camel_1_4	3	3	0	2	1	0	1	2	3	2	1	0	4	5	2	6	1	0	4	0
Camel_1_6	3	2	0	4	0	3	4	2	2	2	1	4	3	2	0	4	1	3	3	3
Jedit_3_4	6	4	1	2	2	2	2	1	2	1	2	5	0	2	3	1	7	4	4	1
Jedit_4_0	2	4	2	1	0	2	0	2	4	2	0	5	2	3	0	2	4	1	2	3
Jedit_4_2	3	2	2	3	3	0	5	1	3	3	1	2	4	1	3	3	4	1	1	4
Jedit_4_3	2	4	1	2	1	0	2	3	3	3	0	6	5	1	1	1	3	1	1	3
log4j_1_0	4	1	2	3	6	1	5	1	8	1	0	1	2	2	0	4	0	3	3	1
log4j_1_1	5	0	3	0	7	3	4	1	6	1	2	1	2	3	0	4	1	2	0	2
log4j_1_2	3	4	3	4	2	4	4	2	3	3	1	2	3	2	2	4	6	1	2	2
lucene_2_0	2	1	2	1	6	2	2	4	1	2	1	0	2	4	2	3	5	4	2	2
lucene_2_2	6	2	1	2	3	2	1	1	6	2	1	3	1	1	3	2	1	3	1	1
lucene_2_4	1	4	3	6	6	2	5	4	4	3	1	1	3	2	2	5	1	3	2	2
xalan_2_4	1	5	2	2	1	0	5	4	2	1	6	2	1	1	2	3	2	1	0	2
xalan_2_5	1	1	0	3	2	0	2	4	4	4	2	0	0	5	1	1	3	2	1	3
xalan_2_6	2	1	3	4	2	0	2	3	3	4	2	0	2	4	3	1	6	3	0	0
xalan_2_7	3	3	1	1	5	3	1	4	2	1	3	2	1	3	1	0	3	6	2	1

- npm, cbm, ic, wmc, rfc, ca are the most selected features and all of them are OO metrics.

Chapter 6

Conclusion and Future Direction

In the previous chapters, a set of SFP models have been investigated to predict the faulty modules in software projects. Different SFP models results has been compared and analyzed. In this chapter, a conclusion of this research work and future direction and recommendations.

6.1 Conclusion

Traditional software QA methods are time consuming and error-prone [16]. That increases the importance of fault prediction models, that highlights faulty modules to be more focused on while doing QA activities and helps in finding potential bugs in early stages. The earlier the bugs caught the lower the cost to fix them, this cost rises in an exponential manner with the time and stage of catching those bugs in SDLC and also catching bugs in earlier releases enhance the overall product quality in later releases.

SFP models varies from depending on software metrics to ML and soft computing [63]. Metrics based models depends on a set of predefined metrics to predict faults. Metrics based models suffers from getting stuck in

detecting bugs in same areas of software for same dataset as well as for other datasets [82].

ML based models outperformed traditional models in extracting knowledge from data even if data is imprecise and/or incomplete and so in classifying software modules as faulty and non-faulty [53]. This research used three different ML algorithms as classifiers which (DT, ANN, kNN) and one as evaluator (L-RNN).

The main challenge that face the ML algorithms is the high dimensionality. FS is one of the dimensionality reduction techniques that proved its ability to improve the performance of different learning algorithms [51]. The latest FS methods in the literature use metaheuristics algorithms as searching strategies to determine the most important features in a dataset. Due to the fact that there is no method that show superior performance for all optimization problems, this research proposed the use of the well known SI algorithms (i.e., PSO and ACO), and an Evolutionary Algorithm (i.e., GA) with the well known classification algorithms (i.e., KNN, DT and ANN) to get the best combination that could select the most informative features that yield the highest classification accuracy in the L-RNN model which is used in the validation phase.

In this research, new FS selection method introduced based on majority voting technique. Majority voting technique depends on taking the most selected features in 9 different combinations of search algorithms and classifiers. Number of features to be taken is the average number of features that selected in the 9 solutions from the 9 combinations.

The experiments conducted in this research shows that majority voting

techniques outperformed all other 9 FS methods with minimum 10% increase in accuracy with average AUC 0.95% when running on 19 datasets, for many of the datasets the classification accuracy was 100%. Moreover, majority technique is promising when it comes to re-usability that is reusing the same set of selected features for dataset in predicting the faults of other datasets.

This research compared the results of running L-RNN model without FS, with FS using the 9 combinations and majority voting technique. The experiments shows that the best FS combination is GA-ANN with average AUC 0.85, the best search algorithm to work with all three classifiers is ACO, ACO tends to choose more features that PSO and GA with 7 selected features on average.

The best classifier to work with all three search algorithms is ANN. However, the worst FS combination is GA-DT. Our experiments shows that the highest features that got selected are all OO metrics namely npm, cbm, ic, wmc, rfc, ca.

6.2 Future Direction

As future directions we need to tackle the performance problem of running all combinations before deciding the best set of features, techniques like incremental majority voting may be experimented. The current majority solution needs all other combinations solution to be calculated first which result in long run time. We also recommend the exploration of general solution existence based on majority voting. In this research the majority voting solution has been calculated for each dataset, the goal is to reach for a solution that can be used as predefined solution for new datasets with similar

characteristics. Another important research area that we highly recommend studying it, is where to inject SFP models in SDLC in real environments, in which phase will it be more effective, and will it really reduce the cost of software QA in such environments. We also recommend running the majority technique in more datasets with different characteristics, bigger in size and in from industrial environments.

Bibliography

[1] Hojjat Adeli and Shih-Lin Hung. Machine learning: neural networks, genetic algorithms, and fuzzy systems. John Wiley & Sons, Inc., 1994.

[2] Mehdi Hosseinzadeh Aghdam, Nasser Ghasem-Aghaee, and Mohammad Ehsan Basiri. "Text feature selection using ant colony optimization". In: Expert systems with applications 36.3 (2009), pp. 6843–6853.

[3] Hamoud I Aljamaan and Mahmoud O Elish. "An empirical study of bagging and boosting ensembles for identifying faulty classes in object-oriented software". In: Computational Intelligence and Data Mining, 2009. CIDM'09. IEEE Symposium on. IEEE. 2009, pp. 187–194.

[4] Ahmed Al-Ani. "Feature subset selection using ant colony optimization". In: International journal of computational intelligence (2005).

[5] Erik Arisholm, Lionel C Briand, and Eivind B Johannessen. "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models". In: Journal of Systems and Software 83.1 (2010), pp. 2–17.

[6] Kent Beck et al. "Manifesto for agile software development". In: (2001).

[7] Rafael Bello et al. "Two-step particle swarm optimization to solve the feature selection problem". In: Intelligent Systems Design and Applications, 2007. ISDA 2007. Seventh International Conference on. IEEE. 2007, pp. 691–696.

[8] David Binkley et al. “Software fault prediction using language processing”. In: Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION, 2007. TAICPART-MUTATION 2007. IEEE. 2007, pp. 99–110.

[9] Christian Blum. “Ant colony optimization: Introduction and recent trends”. In: Physics of Life reviews 2.4 (2005), pp. 353–373.

[10] Fletcher J Buckley and Robert Poston. “Software quality assurance”. In: IEEE Transactions on Software Engineering 1 (1984), pp. 36–41.

[11] Jaspar Cahill, James M Hogan, and Richard Thomas. “Predicting fault-prone software modules with rank sum classification”. In: Software Engineering Conference (ASWEC), 2013 22nd Australian. IEEE. 2013, pp. 211–219.

[12] Gabriella Carrozza et al. “Analysis and prediction of mandelbugs in an industrial software system”. In: Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on. IEEE. 2013, pp. 262–271.

[13] Cagatay Catal. “Software fault prediction: A literature review and current trends”. In: Expert systems with applications 38.4 (2011), pp. 4626–4636.

[14] Cagatay Catal and Banu Diri. “A fault prediction model with limited fault data to improve test process”. In: International Conference on Product Focused Software Process Improvement. Springer. 2008, pp. 244–257.

[15] Cagatay Catal and Banu Diri. “Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem”. In: Information Sciences 179.8 (2009), pp. 1040–1058.

[16] Venkata UB Challagulla, Farokh B Bastani, and I-Ling Yen. “A unified framework for defect data analysis using the mbr technique”. In: null. IEEE. 2006, pp. 39–46.

[17] Girish Chandrashekar and Ferat Sahin. “A survey on feature selection methods”. In: Computers & Electrical Engineering 40.1 (2014), pp. 16–28.

[18] Ching-Pao Chang, Chih-Ping Chu, and Yu-Fang Yeh. “Integrating in-process software defect prediction with association mining to discover defect pattern”. In: Information and software technology 51.2 (2009), pp. 375–384.

[19] Shyam R Chidamber and Chris F Kemerer. “A metrics suite for object oriented design”. In: IEEE Transactions on software engineering 20.6 (1994), pp. 476–493.

[20] Shyam R Chidamber and Chris F Kemerer. Towards a metrics suite for object oriented design. Vol. 26. 11. ACM, 1991.

[21] Marco D’Ambros, Michele Lanza, and Romain Robbes. “An extensive comparison of bug prediction approaches”. In: Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on. IEEE. 2010, pp. 31–41.

[22] Silvia N das Dôres et al. “A meta-learning framework for algorithm recommendation in software fault prediction”. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing. ACM. 2016, pp. 1486–1491.

[23] Marco Dorigo and Mauro Birattari. “Ant colony optimization”. In: Encyclopedia of machine learning. Springer, 2011, pp. 36–39.

[24] Marco Dorigo and Gianni Di Caro. “Ant colony optimization: a new meta-heuristic”. In: Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on. Vol. 2. IEEE. 1999, pp. 1470–1477.

[25] Marco Dorigo and Luca Maria Gambardella. “Ant colonies for the travelling salesman problem”. In: biosystems 43.2 (1997), pp. 73–81.

[26] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. “Ant system: optimization by a colony of cooperating agents”. In: IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 26.1 (1996), pp. 29–41.

[27] Russell C Eberhart and Yuhui Shi. “Comparison between genetic algorithms and particle swarm optimization”. In: International conference on evolutionary programming. Springer. 1998, pp. 611–616.

[28] Ezgi Erturk and Ebru Akcapinar Sezer. “Iterative software fault prediction with a hybrid approach”. In: Applied Soft Computing 49 (2016), pp. 1020–1033.

[29] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. “From data mining to knowledge discovery in databases”. In: AI magazine 17.3 (1996), p. 37.

[30] Robert Firth et al. A guide to the classification and assessment of software engineering tools. Tech. rep. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 1987.

[31] Daniel Galin. Software quality assurance: from theory to implementation. Pearson Education India, 2004.

[32] David E Goldberg and John H Holland. “Genetic algorithms and machine learning”. In: Machine learning 3.2 (1988), pp. 95–99.

[33] Isabelle Guyon and André Elisseeff. “An introduction to variable and feature selection”. In: Journal of machine learning research 3.Mar (2003), pp. 1157–1182.

[34] Tibor Gyimothy, Rudolf Ferenc, and Istvan Siket. “Empirical validation of object-oriented metrics on open source software for fault prediction”. In: IEEE Transactions on Software engineering 31.10 (2005), pp. 897–910.

[35] Tracy Hall et al. “A systematic literature review on fault prediction performance in software engineering”. In: IEEE Transactions on Software Engineering 38.6 (2012), pp. 1276–1304.

[36] Maurice H Halstead et al. Elements of Software Science (Operating and programming systems series). Elsevier Science Inc., New York, NY, 1977.

[37] Georgef Hepner et al. “Artificial neural network classification using a minimal training set- Comparison to conventional supervised classification”. In: Photogrammetric Engineering and Remote Sensing 56.4 (1990), pp. 469–473.

[38] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. Applied logistic regression. Vol. 398. John Wiley & Sons, 2013.

[39] George H. John, Ron Kohavi, and Karl Pfleger. “Irrelevant Features and the Subset Selection Problem”. In: MACHINE LEARNING: PROCEEDINGS OF THE ELEVENTH INTERNATIONAL. Morgan Kaufmann, 1994, pp. 121–129.

[40] Anum Kalsoom et al. “A dimensionality reduction-based efficient software fault prediction using Fisher linear discriminant analysis (FLDA)”. In: The Journal of Supercomputing (2018), pp. 1–35.

[41] Shima Kashef and Hossein Nezamabadi-pour. “An advanced ACO algorithm for feature subset selection”. In: Neurocomputing 147 (2015), pp. 271–279.

[42] Gopalan Kesavaraj and Sreekumar Sukumaran. “A study on classification techniques in data mining”. In: Computing, Communications and Networking Technologies (ICCCNT), 2013 Fourth International Conference on. IEEE. 2013, pp. 1–7.

[43] Taghi M Khoshgoftaar, Erik Geleyn, and Laruent Nguyen. “Empirical case studies of combining software quality classification models”. In: Quality Software, 2003. Proceedings. Third International Conference on. IEEE. 2003, pp. 40–49.

[44] Barbara A Kitchenham. “Software quality assurance”. In: Microprocessors and microsystems 13.6 (1989), pp. 373–381.

[45] Ron Kohavi and George H John. “Wrappers for feature subset selection”. In: Artificial intelligence 97.1-2 (1997), pp. 273–324.

[46] A Güneş Koru et al. “Theory of relative defect proneness”. In: Empirical Software Engineering 13.5 (2008), p. 473.

[47] Stefan Lessmann et al. “Benchmarking classification models for software defect prediction: A proposed framework and novel findings”. In: IEEE Transactions on Software Engineering 34.4 (2008), pp. 485–496.

[48] Hareton KN Leung and Lee White. “Insights into regression testing (software testing)”. In: Software Maintenance, 1989., Proceedings., Conference on. IEEE. 1989, pp. 60–69.

[49] Zachary C Lipton, John Berkowitz, and Charles Elkan. “A critical review of recurrent neural networks for sequence learning”. In: arXiv preprint arXiv:1506.00019 (2015).

[50] Huan Liu and Hiroshi Motoda. Feature extraction, construction and selection: A data mining perspective. Vol. 453. Springer Science & Business Media, 1998.

[51] Huan Liu and Hiroshi Motoda. Feature selection for knowledge discovery and data mining. Vol. 454. Springer Science & Business Media, 2012.

[52] M Mahalakshmi and M Sundararajan. "Traditional SDLC Vs Scrum Methodology– A Comparative Study". In: International Journal of Emerging Technology and Advanced Engineering 3.6 (2013), pp. 192–196.

[53] Ruchika Malhotra. "A systematic review of machine learning techniques for software fault prediction". In: Applied Soft Computing 27 (2015), pp. 504–518.

[54] Rammohan Mallipeddi et al. "Differential evolution algorithm with ensemble of parameters and mutation strategies". In: Applied soft computing 11.2 (2011), pp. 1679–1696.

[55] Alianna J Maren, Craig T Harston, and Robert M Pap. Handbook of neural computing applications. Academic Press, 2014.

[56] Thilo Mende and Rainer Koschke. "Revisiting the evaluation of defect prediction models". In: Proceedings of the 5th International Conference on Predictor Models in Software Engineering. ACM. 2009, p. 7.

[57] Tim Menzies, Jeremy Greenwald, and Art Frank. "Data mining static code attributes to learn defect predictors". In: IEEE transactions on software engineering 33.1 (2007), pp. 2–13.

[58] Ayşe Tosun Mısırlı, Ayşe Başar Bener, and Burak Turhan. "An industrial case study of classifier ensembles for locating software defects". In: Software Quality Journal 19.3 (2011), pp. 515–536.

[59] Chris Murphy. "Is Data Mining Free Speech?" In: InformationWeek (2011).

[60] Glenford J Myers, Corey Sandler, and Tom Badgett. The art of software testing. John Wiley & Sons, 2011.

[61] Dawn M Owens and Deepak Khazanchi. “Software quality assurance”. In: Handbook of Research on Technology Project Management, Planning, and Operations. IGI Global, 2009, pp. 242–260.

[62] Danijel Radjenović et al. “Software fault prediction metrics: A systematic literature review”. In: Information and Software Technology 55.8 (2013), pp. 1397–1418.

[63] Santosh S Rathore and Sandeep Kumar. “A decision tree logic based recommendation system to select software fault prediction techniques”. In: Computing 99.3 (2017), pp. 255–285.

[64] P Ashok Reddy, K Rajasekhara Rao, and M Babu Reddy. “Performance evaluation of procedural metrics and object oriented metrics”. In: International Journal of Research Studies in Computer Science and Engineering 2.3 (2015), pp. 69–72.

[65] J. Sayyad Shirabad and T.J. Menzies. The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada. 2005. url: http://promise.site.uottawa.ca/SERepository.

[66] Gordon Gordon Schulmeyer and James I McManus. Handbook of software quality assurance. Van Nostrand Reinhold Co., 1992.

[67] Raed Shatnawi. “The application of ROC analysis in threshold identification, data imbalance and metrics selection for software fault prediction”. In: Innovations in Systems and Software Engineering 13.2-3 (2017), pp. 201–217.

[68] Martin Shepperd et al. NASA MDP Software Defects Data Sets. Mar. 2018. doi: 10.6084/m9.figshare.c.4054940.v1. url: https://figshare.com/collections/NASA_MDP_Software_Defects_Data_Sets/4054940/1.

[69] David J Sheskin. Handbook of parametric and nonparametric statistical procedures. crc Press, 2003.

[70] Qinbao Song et al. “A general software defect-proneness prediction framework”. In: IEEE Transactions on Software Engineering 37.3 (2011), pp. 356–370.

[71] Jerffeson Souza, Nathalie Japkowicz, and Stan Matwin. “Feature selection with a general hybrid algorithm”. In: Feature Selection for Data Mining (2005), p. 45.

[72] E-G Talbi. “A taxonomy of hybrid metaheuristics”. In: Journal of heuristics 8.5 (2002), pp. 541–564.

[73] El-Ghazali Talbi. Metaheuristics: from design to implementation. Vol. 74. John Wiley & Sons, 2009.

[74] Pang-Ning Tan et al. Introduction to data mining. Pearson Education India, 2006.

[75] Ayşe Tosun, Burak Turhan, and Ayşe Bener. “Validation of network measures as indicators of defective modules in software systems”. In: Proceedings of the 5th international conference on predictor models in software engineering. ACM. 2009, p. 5.

[76] Hamza Turabieh, Majdi Mafarja, and Xiaodong Li. “Iterated feature selection algorithms with layered recurrent neural network for software fault prediction”. In: Expert Systems with Applications 122 (2019), pp. 27–42.

[77] Bhekisipho Twala. “Software faults prediction using multiple classifiers”. In: Computer Research and Development (ICCRD), 2011 3rd International Conference on. Vol. 4. IEEE. 2011, pp. 504–510.

[78] David H Wolpert and William G Macready. “No free lunch theorems for optimization”. In: IEEE transactions on evolutionary computation 1.1 (1997), pp. 67–82.

[79] Bingbing Yang et al. “Software quality prediction using affinity propagation algorithm”. In: Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on. IEEE. 2008, pp. 1891–1896.

[80] Amin Zarshenas and Kenji Suzuki. “Binary coordinate ascent: An efficient optimization technique for feature subset selection for machine learning”. In: Knowledge-Based Systems 110 (2016), pp. 191–201.

[81] Feng Zhang et al. “Towards building a universal defect prediction model with rank transformed predictors”. In: Empirical Software Engineering 21.5 (2016), pp. 2107–2145.

[82] Thomas Zimmermann, Nachiappan Nagappan, and Andreas Zeller. “Predicting bugs from history”. In: Software Evolution. Springer, 2008, pp. 69–88.