LEAST SQUARE SUPPORT VECTOR MACHINE(LSSVM) FOR SOFTWARE DEFECT FORECASTING

Dissertation submitted in partial fulfillment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

By

S.SAI LIKHITH YADAV	**16VV1A1244**
A.VINEELA	**16VV1A1201**
E.JYOTHSNA	**16VV1A1209**
T.SAMPATH SRIVATHSAV	**15VV1A1201**

Under the esteemed Guidance of

Mr. B. TIRIMULA RAO

Assistant Professor

Department of Information Technology

DEPARTMENT OF INFORMATION TECHNOLOGY

JNTUK-University College of Engineering Vizianagaram

VIZIANAGARAM-535003, A.P., INDIA

APRIL, 2020

Department Of Information Technology

JNTUK-University College of Engineering Vizianagaram

VIZIANAGARAM-535003, A.P., INDIA

CERTIFICATE

This is to certify that the dissertation entitled "**LEAST SQUARE SUPPORT VECTOR MACHINE (LSSVM) FOR SOFTWARE DEFECT FORECASTING**" submitted by **S. SAI LIKHITH YADAV(16VV1A1244), A.VINEELA(16VV1A1201), E.JYOTHSNA(16VV1A1209), T. SAMPATH SRIVATHSAV(15VV1A1201),** in partial fulfilment for the award of the degree of **BACHELOR OF TECHNOLOGY** in **INFORMATION TECHNOLOGY** from Jawaharlal Nehru Technological University Kakinada-University College of Engineering Vizianagaram is a record of bonafide work carried out by them under my guidance and supervision during the year 2019-2020. The results embodied in this dissertation have not been submitted by any other University or Institution for the award of any degree.

Signature of the Guide

B.TIRIMULA RAO

Assistant Professor

Dept. of Information Technology

JNTUK-UCEV, VIZIANAGARAM

Signature of the Head of the Department

Prof. G.JAYASUMA

Professor & HOD

Dept. of Information Technology

JNTUK-UCEV,VIZIANAGARAM

DECLARATION

We, S. Sai Likhith Yadav(16VV1A1244), A.Vineela(16VV1A1201), E.Jyothsna(16VV1A1209), T. Sampath Srivathsav (15VV1A1201), declare that the project report entitled "**LEAST SQUARE SUPPORT VECTOR MACHINE (LSSVM) FOR SOFTWARE DEFECT FORECASTING"** is no more than 1,00,000 words in length including quotes and exclusive of tables, figures, bibliography and reference. This dissertation contains no material that has been submitted previously, in whole or in part, for the award of any other academic degree or diploma. Except where otherwise indicated this project is our own work.

Roll no	Name	Signature
16VV1A1244	S.Sai Likhith Yadav	----------
16VV1A1201	A.Vineela	----------
16VV1A1209	E. Jyothsna	----------
15VV1A1201	T. Sampath Srivathsav	----------

Date:

ACKNOWLEDGEMENT

This project report could not have been written without the support of our professor **Mr.B.Tirimula Rao, Assistant Professor, Information Technology** department who not only served as our superior but also encouraged and challenged us throughout our academic program. Our foremost gratitude goes to him. Without him, this project would not have been possible. We appreciate his vast knowledge in many areas and his insights, suggestions and guidance that helped to shape our research skills.

It is needed with a great sense of pleasure and an immense sense of gratitude that we acknowledge the help of these individuals. We owe many thanks to many people who helped and supported us during the writing of this report.

We are thankful to our project coordinator **Mr. W. Anil, Assistant Professor,**Department of Information Technology, for his continuous support.

We express our sincere thanks to our respected **Prof. G.Jaya Suma, H.O.D of Information Technology** of JNTUK University College of Engineering Vizianagaram for her valuable suggestion and constant motivation that greatly helped us in the successful completion of the project. We also take the privilege to express our heartfelt gratitude to **Prof. G.Swami Naidu, Principal**, JNTUK University College of Engineering.

We are thankful to all faculty members for extending their kind cooperation and assistance. Finally, we are extremely thankful to our parents and friends for their constant help and moral support.

S. SAI LIKHITH YADAV **(16VV1A1244)**

A.VINEELA **(16VV1A1201)**

E. JYOTHSNA **(16VV1A1209)**

T. SAMPATH SRIVATHSAV **(15VV1A1201)**

TABLE OF CONTENTS

ABSTRACT

Least Square Support Vector Machine (LSSVM) for Software Defect Forecasting

In today's world, software products have been an important resource used by various organizations which are indeed to be error-free and easily accessible. A fault is a condition that causes the software to fail to perform its required function. Identification and fixing of faults pose a greater challenge for the software developers reported by testers and users. This provides an impact in terms of saving maintenance resources, user satisfaction and averting major system failures post-deployment.

Least Square Support Vector Machine(LSSVM) is the least square version of Support Vector Machine(SVM) which is based on the theory of statistical learning and its main goal is to provide optimization. LSSVM provides a suitable baseline model for building fault predictive tool as this method is associated with various kernel functions like Linear, Polynomial and Radial basis function. Here, the kernel functions enable the dot product to be performed in high dimensional feature space using low dimensional space data.

The work presented in this paper involves building a defect prognosis tool by analyzing and examining the predictive power of software metrics. We apply various feature selection techniques (i.e. feature subset selection and feature ranking methods) to choose the best set of metrics from a given data set and reduces misclassification errors. Least Squares Support Vector Machine(LSSVM) is the approach on which the fault prediction model is framed. This model as a classifier predicts whether a class is faulty or not. These metrics are then validated using cost analysis model.

DOMAIN: Supervised Machine Learning.

KEYWORDS: Cost analysis, Fault, Feature selection techniques, Least Squares Support Vector Machine(LSSVM).

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

INTRODUCTION

In today's world, software products have been an important resource used by various organisations which are indeed to be error-free and easily accessible. A fault is a condition that causes the software to fail to perform its required function. Identification and fixing of faults pose a greater challenge for the software developers reported by testers and users. This provides an impact in terms of saving maintenance resources, user satisfaction and averting major system failures post-deployment.

1.1 Problem Statement

The main problem is to build a model using Least Square Support Vector Machine for several classes in an Object-Oriented Software Sub-Systems (represented in datasets) and predict whether the class is faulty or not based on the performance metrics.

1.2 Motivation of LS-SVM

The Least Square SVM(LSSVM) is the least-square variant in Support Vector Machine which is based on the theory of statistical learning. There are many applications of LSSVM, such as data analyzation and pattern recognition which are used for classification and regression analysis.

The main goal of LSSVM is to provide optimization. LSSVM provides a suitable baseline model for building fault predictive tool as this method is associated with various kernel functions like Linear, Polynomial and Radial basis function. Here, the kernel functions enable the dot product to be performed in high dimensional feature space using low dimensional space data.

CHAPTER 2
REVIEW ON LITERATURE

REVIEW ON LITERATURE

2.1 Emergence of Least-Square SVM

For the reduction of the computational complexity of SVM, Suykens has proposed the least-square version of SVM known as Least Square Support Vector Machine(LSSVM). In LSSVM, the inequality constraints are replaced with equality constraints in solving quadratic programming. This results in faster speed in the training process compared to SVM. Nonetheless, lack of sparseness and robustness in the computational solution are the two main drawbacks of LSSVM. These drawbacks will increase the training time and reduce the accuracy of a predicted model for the real industrial data sets, which have intractable characters such as imbalanced distribution, heteroscedasticity(circumstance in which the variability of a variable is unequal across the range of values of a second variable that predicts it.), and the explosion of data. Therefore, this work focuses on the optimization of LSSVM and usage of various kernels as classifiers for developing a prototype to classify faulty and non-faulty classes.

2.2 Least Square -Support Vector Machine:

In this work, Least Square Support Vector Machine (LSSVM) with various kernels have been considered as a classifier for developing a model to classify faulty and non faulty classes.

The general form of LSSVM function is defined as:

$$y(x) = w^{T} \cdot \Phi(x) + c$$

where , y and x are the output and input vector. $\varphi(x)$ is nonlinear mapping function and it is used for mapping the input data into higher dimensional feature space. w , and c show the adjust weight vector and scalar threshold value respectively.

The objective in LSSVM is to optimize the following equation:

$$\text{Minimize } \frac{1}{2} w^{T} w + \gamma \frac{1}{2} \sum_{(i=1)}^{1} E_i^2$$

$$\text{Subject w.r.t } y(x) = w^{T} \Phi(x) + c + E_i, i = 1,2,3,\ldots n$$

where E_i is the error value of input instance i and γ is the cost function.

After solving the above optimization problem, fault prediction values are obtained using the below equation:

$$Y' = \sum_{(i=1)}^{l} (\alpha_i - \alpha_i^*)\Phi(x_i) * \Phi(x) + b = \sum_{(i=1)}^{l} (\alpha_i - \alpha_i^*) * K(x_i, x) + b$$

where $K(x_i, x)$ is the kernel function, which enables the dot product to be performed in high-dimensional feature space using low- dimensional space data.

LS-SVM algorithms use a set of mathematical functions that are defined as the kernel which separates the inseparable data with increasing dimensionality. In simple terms, the function of the kernel is to take data as input and transform it into the required form. The kernel function plays the role of a dot product between the normal vector and the input vector in the higher dimensional feature space. Different SVM algorithms use different types of kernel functions. These functions can be of different types. For example Linear, Non-Linear, Polynomial, Radial basis function (RBF), and Sigmoid.

The different Kernel functions considered in this study are mentioned below:

1. Linear function
2. Polynomial function
3. Radial basis function

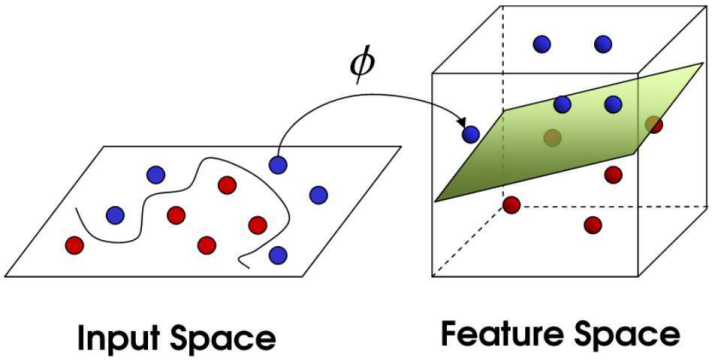

Fig 2.1: Representation of functionality of kernel method

The linear, polynomial and RBF or Gaussian kernel are simply different in case of making the hyperplane decision boundary between the classes. The kernel functions are used to map the original dataset (linear/nonlinear) into a higher dimensional space with a view to making it linear dataset.

Usually linear and polynomial kernels are less time consuming and provide less accuracy than the RBF or Gaussian kernels. The k cross-validation is used to divide the training set into k distinct subsets. Then every subset is used for training and other k-1 are used for validation in the entire training phase. This is done for the better training of the classification task.

2.3 Optimization of LSSVM

The objective in LSSVM is to optimize the following equation:

$$\min_{(w,e)} \tau(w,e) = \frac{1}{2} w^T w + \gamma \frac{1}{2} \sum_{(k=1)}^{N} e_k^2$$

subject w.r.t equality constraints

$$y_k = w^T \varphi(x_k) + b + e_k \,, k = 1,2,\ldots, N$$

where e_k is the error value of input instance i and φ is the cost function.

To solve this problem, we could construct the Lagrangian function:

$$L(w,b,e;\alpha) = \tau(w,e) - \sum_{(k=1)}^{N} \alpha_k \{ w^T \varphi(x_k) + b + e_k - y_k \}$$

where α_k refered as Lagrangian multipliers.

The conditions for optimality are:

$$\frac{\partial L}{\partial w} = 0 \rightarrow w = \sum_{(k=1)}^{N} \alpha_k \varphi(x_k)$$

$$\frac{\partial L}{\partial b} = 0 \rightarrow \sum_{(k=1)}^{N} \alpha_k = 0$$

$$\frac{\partial L}{\partial e_k} = 0 \rightarrow \alpha_k = \gamma e_k \quad k = 1,2,\ldots, N$$

$$\frac{\partial L}{\partial \alpha_k} = 0 \rightarrow w^T \varphi(x_k) + b + e_k - y_k = 0 \quad k = 1,2,\ldots, N$$

Elimination of *w* and *e* will yield a linear system instead of a quadratic programming problem:

$$\begin{bmatrix} 0 & \vec{1}^{T} \\ \vec{1} & \Omega + \gamma^{-1} I \end{bmatrix} \begin{bmatrix} b \\ \vec{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ y \end{bmatrix} \text{ with } y = [y_1; \ldots; y_N], \vec{1} = [1; \ldots; 1], \alpha = [\alpha_1; \ldots; \alpha_N]$$

Here, $\mathbf{I_N}$ is an N x N identity matrix for the kernel matrix $\boldsymbol{\Omega}$

Applying Mercer's condition is given as:

$$\begin{aligned} \Omega_{(k,l)} &= \varphi(x_k)^{T} \varphi(x_l), \quad k, l = , \ldots . , N \\ &= K(x_k, x_l) \end{aligned}$$

$$\text{where } K(x_k, x_l) \text{ is the kernel function.}$$

So, the resulting LS-SVM model for function estimation is given as:

$$y(x) = \sum_{(k=1)}^{N} \alpha_k \; K(x_k, x) + b$$

2.4 Comparisons of SVM and LS-SVM

Least-square SVM is the standard approach to approximate the solution of overdetermined systems when compared to normal SVM. The hard margin SVM which is based on the Euclidean distance measure may be comparable to LS-SVM for high dimensional small sample-sized data. Another difference is that SVM requires you to solve a quadratic programming problem while LS-SVM requires you to solve a linear system as LS-SVM is derived from Ridge Regression which has the same functional form as of SVM. For inseparable data, LS-SVM is preferable to normal SVM.

2.5 Dataset

A data set (or dataset) is a collection of data. Most commonly a data set corresponds to the contents of a single database table, or a single statistical data matrix, where every column of the table represents a particular variable, and each row corresponds to a given member of the data set in question. The data set lists values for each of the variables, such as height and weight of an object, for each member of the data set. Each value is known as a datum. The data set may comprise data for one or more members, corresponding to the number of rows. Data sets that are so large that traditional data processing applications are inadequate to deal with them are known as big data.

CHAPTER 3

SOFTWARE REQUIREMENT SPECIFICATION DOCUMENT

SOFTWARE REQUIREMENT SPECIFICATION DOCUMENT

In this section, we analyse and briefly discuss the purpose of this project followed by a general procedure and specification constraint based on acceptance criteria.

3.1 Introduction

3.1.1 Purpose

The purpose of this project is to use the Least-Square Support Vector Machine concept with various kernels for classification. Least Square Support Vector Machine (LSSVM) with various kernels and the performance metric-based cost model has been considered as a classifier for developing a model to classify faulty and non-faulty classes.

3.1.2 Scope

Create a user benefited environment with related permissions. It helps to maintain fully organized dataset information, which will facilitate an easy observation of the process by the users.

3.2 Functions

- Loading dataset
- Applying Feature Selection techniques
- Applying Normalisation
- Applying LS-SVM function across three kernels(Linear,Polynomial and Radial-basis)
- Performing training and testing using cross-validation approach.
- Generation of classification report and confusion matrix
- Calculation of Normalized Cost for the obtained matrix for selected cost parameter
- Classification based on the normalized cost

3.2.1 User Characteristics

The user is supposed to have the following with him/her

- A dataset, in a format supported by the software
- The user must know what is meant by Least-Square SVM with the associated kernels and concept of confusion matrix so that he/she should be able to compare predicted and actual values for the obtained results.

3.2.2 General Constraints

The system should be running on Windows or Linux and should be provided with the R studio environment and Python programming environment aided with required packages.

3.3 Specific Requirements

3.3.1 Software Requirements

3.3.1.1 R Studio

RStudio is a free and open-source integrated development environment (IDE) for R, a programming language for statistical computing and graphics. RStudio was founded by JJ Allaire, creator of the programming language ColdFusion. Hadley Wickham is the Chief Scientist at RStudio.

RStudio is available in two editions: RStudio Desktop, where the program is run locally as a regular desktop application; and RStudio Server, which allows accessing RStudio using a web browser while it is running on a remote Linux server. Pre-packaged distributions of RStudio Desktop are available for Windows, MacOS, and Linux.

Packages required in R are:

- RoughSets -for Rough Set Analysis
- Fselector -for Feature Selection Methods
- Metrics and other packages

3.3.1.2 Python Environment:The Jupyter Notebook

Jupyter Notebook (formerly IPython Notebooks) is a web-based interactive computational environment for creating Jupyter notebook documents.

A Jupyter Notebook document is a JSON document, following a versioned schema, and containing an ordered list of input/output cells which can contain code, text (using Markdown), mathematics, plots and rich media, usually ending with the ".ipynb" extension.

Packages required in Python are:

- sklearn (Scikit-learn) -for various classification, regression and clustering algorithms
- SciPy -for scientific computing and technical computing
- NumPy - for multi-dimensional arrays and matrices
- pandas - for data manipulation and analysis
- statistics and other package

3.3.1.3 Waikato Environment for Knowledge Analysis (WEKA)

WEKA - an open-source software provides tools for data preprocessing, implementation of several Machine Learning algorithms, and visualization tools so that a user can develop machine learning techniques and apply them to real-world data mining problems.

Weka supports several standard data mining tasks, more specifically, data preprocessing, clustering, classification, regression, visualization, and feature selection. All of Weka's techniques are predicated on the assumption that the data is available as one flat file or relation, where each data point is described by a fixed number of attributes (normally, numeric or nominal attributes, but some other attribute types are also supported).

Weka provides access to SQL databases using Java Database Connectivity and can process the result returned by a database query. Weka provides access to deep learning with Deeplearning4j.

Packages required in Weka are:

- normalize- for preprocessing
- chiSquaredAttributeEval -for attribute selection
- consistencySubsetEval -for attribute selection
- filteredAttributeSelection - for attribute selection
- weka .attributeSelection - for attribute selection(inbuilt) (consists of GainRatioAttributeEval,CfsAttributeEval,InfoGainAttributeEval,OneRAttributeEval, PrincipalComponents)

3.3.1.4 Operating System

- Windows/Linux operating system

3.3.2 Hardware Requirements

- Minimum of 1GB Hard Disk
- Minimum of 2048MB RAM

CHAPTER 4
METHODOLOGY

METHODOLOGY

4.1 Learning-Based Model

Learning-Oriented techniques include both some of the oldest as well as the newest techniques applied to prediction activities. They attempt to automate improvements in the estimation process by building models that “Learn” from previous experience. Here, LS-SVM is a class of kernel-based learning methods.

4.2 Classification Techniques

We have several classification techniques available. Classification analysis is the supervised process of assigning items to categories/classes in order to improve the accuracy of the model. In classification, we use the training dataset to get better boundary conditions which could be used to determine each target class. Once the boundary conditions are determined, the next task is to predict the target class. In this work, the utilization of Least-Square SVM over the traditional SVM method allows minimizes the sum of squared errors to the objective function. SVM requires you to solve a quadratic programming problem while LS-SVM requires you to solve a linear system.

4.3 Cross Validation

Cross-validation, sometimes called rotation estimation, is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. In a prediction problem, a model is usually given a dataset of known data on which training is run (training dataset), and a dataset of unknown data (or first seen data) against which the model is tested (testing dataset).

The goal of cross-validation is to define a dataset to "test" the model in the training phase (i.e., the validation dataset), in order to limit problems like overfitting, give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem), etc.

One of the main reasons for using cross-validation instead of using the conventional validation (i.e. partitioning the data set into two sets of 80% for training and 20% for testing) is that there is not enough data available to partition it into separate training and test sets without losing significant modeling or testing capability. In these cases, a fair way to properly estimate model prediction performance is to use cross-validation as a powerful general technique.

In summary, cross-validation combines (averages) measures of fit (prediction error) to derive a more accurate estimate of model prediction performance.

4.3.1 Common types of cross validation

Two types of cross-validation can be distinguished, exhaustive and non-exhaustive cross-validation.

4.3.1.1 Exhaustive cross-validation

Exhaustive cross-validation methods are cross-validation methods which learn and test on all possible ways to divide the original sample into training and a validation set.

4.3.1.1.1 Leave-p-out cross-validation:

Leave-p-out cross-validation (LpO CV) involves using p observations as the validation set and the remaining observations as the training set. This is repeated in all ways to cut the original sample on a validation set of p observations and a training set.

4.3.1.1.2 Leave-one-out cross-validation

Leave-one-out cross-validation (LOOCV) is a particular case of leave-p-out cross-validation with p = 1. The process looks similar to jackknife, however with cross-validation, you compute a statistic on the left-out sample(s), while with jack-knifing you compute a statistic from the kept samples only. LOO cross-validation does not have the same problem of excessive compute time as general LpO cross-validation.

4.3.1.2 Non-exhaustive cross-validation

Non-exhaustive cross validation methods do not compute all ways of splitting the original sample. Those methods are approximations of leave-p-out cross-validation.

4.3.2.1 K-fold cross-validation

In k-fold cross-validation, the original sample is randomly partitioned into k equal sized sub-samples. Of the k sub-samples, a single subsample is retained as the validation data for testing the model, and the remaining k − 1 subsample are used as training data. The cross-validation process is then repeated k times (the folds), with each of the k sub-samples used exactly once as the validation data. The k results from the folds can then be averaged to produce a single estimation. The advantage of this method over repeated random sub-sampling (see below) is that all observations are used for both training and validation, and each observation is used for validation exactly once. 10-fold cross-validation is commonly used, but in general, k remains an unfixed parameter.

For example, setting k = 2 results in 2-fold cross-validation. In 2-fold cross-validation, we randomly shuffle the dataset into two sets d_0 and d_1, so that both sets are equal size (this is usually implemented by shuffling the data array and then splitting it in two). We then train on d_0 and test on d_1 , followed by training on d_1 and testing on d_0.

When k = n (the number of observations), the k-fold cross-validation is exactly the leave-one-out cross-validation.

In stratified k-fold cross-validation, the folds are selected so that the mean response value is approximately equal in all the folds. In the case of a dichotomous classification, this means that each fold contains roughly the same proportions of the two types of class labels.

4.4 Normalization

Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information. Normalization is also required for some algorithms to model the data correctly.

For example, assume your input dataset contains one column with values ranging from 0 to 1, and another column with values ranging from 10,000 to 100,000. The great difference in the scale of the numbers could cause problems when you attempt to combine the values as features during modelling. Normalization avoids these problems by creating new values that maintain the general distribution and ratios in the source data while keeping values within a scale applied across all numeric columns used in the model.

Methods of Data Normalization :

- Decimal Scaling
- Min-Max Normalization
- Z-Score Normalization(Zero-Mean Normalization)

4.4.1 Decimal Scaling:

Decimal Scaling is a normalization technique in which we normalize the given value by moving the decimal points of that value. The number of decimal points to move is defined by the maximum absolute value of the given data set. If V_i value of attribute A, then normalized value U_i is given as:

$$U_i = \frac{V_i}{10^j} \quad \text{, where j is the smallest integer such that } \max|U_i| < 1$$

Suppose we have data set in which the value ranges from -9900 to 9877. In this case, the maximum absolute value is 9900. So to perform decimal normalization, we divide each of the values into data set by 10000 (i.e) j = 4.(since it near to 9900).

4.4.2 Z-Score Normalization:

This is a normalization technique in which values are normalized based on mean and standard deviation of the values of the given data set. If z and x are new and old of each entry in dataset respectively and mean() and stdev() are the mean and standard deviation of x respectively. Then,

$$z = \frac{x - \text{mean}(x)}{\text{stdev}(x)}$$

For example, let's say you have a test score of 190. The test has a mean (μ) of 150 and a standard deviation(σ) of 25. Assuming a normal distribution, z score would be:

$$z = (x - \mu) / \sigma$$
$$= (190 - 150) / 25$$
$$= 1.6$$

4.4.3 Min-Max Normalization:

This is a normalization technique in which linear transformation is performed on the original data. Minimum and maximum value from data is fetched and each value is replaced with the existing values of the given data set. Here, the scaling of values is generally done in the range [0,1]. If z and x are the new and old of each entry in dataset respectively and min() and max() are the minimum and maximum of x respectively. Then,

$$z = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Consider the example below, the duration calls value is 50000, we want to transform this in to the range [0.0, 1.0], so first we find the maximum value of duration calls which is 55000 and the minimum value of duration calls, 25000, them the new scaled value for 50000 will be:

$$z = [x - \min(x)] / [\max(x) - \min(x)]$$
$$= (50000 - 25000) / (55000 - 25000)$$
$$= 0.83333333333333337$$

Now let's apply the normalization technique to all the twenty attributes in our data set.
The attributes need to be scaled to fit in the range [0.0, 1.0]. Applying the min-max normalization formula above, we get the normalized data set as given below in table 4.1 and table 4.2. Throughout the implementation is done on all the required datasets. All the datasets have 21 attributes of which the last attribute is the fault named as bug which is a class label. These datasets are pre-processed and applied to the designed technique.

Table 4.1: Normalized Data set after applying min-max normalization for Ant-Class Dataset

wmc	dit	noc	cbo	rfc	lcom	ca
0.025	0	0	0.02	0.0625	0.0004	0.002
0.0417	0.1667	0	0.008	0.0451	0	0.002
0.0083	0.1667	0	0.002	0.0104	0	0
0.0667	0	0.0882	0.0261	0.0694	0.0018	0.0181
0.075	0.3333	0	0.01	0.0903	0.0024	0
0.025	0.1667	0.049	0.014	0.0139	0.0001	0.012
0.1667	0	0	0.008	0.1389	0.0194	0

ce	npm	lcom3	loc	dam	moa	mfa
0.2432	0.0097	0.55	0.0233	0	0	0
0.1081	0.0388	0.3125	0.0167	1	0.0909	0.7
0.027	0.0097	1	0.0015	0	0	1
0.1081	0.0777	0.4	0.0222	0.2	0.0909	0
0.1351	0.068	0.375	0.0407	1	0	0.8
0.027	0.0194	0.25	0.0035	1	0	0.75
0.1081	0.1748	0.3684	0.076	1	0.0909	0

cam	ic	cbm	amc	max_cc	avg_cc	bug
0.4444	0	0	0.0159	0.0189	0.0984	0
0.5	0	0	0.0065	0.0189	0.0885	0
1	0	0	0.0029	0	0	0
0.4063	0	0	0.0054	0.0189	0.1291	0
0.3889	0	0	0.0093	0.0377	0.1475	1
0.5556	0	0	0.0019	0.0189	0.0984	0
0.2842	0	0	0.0077	0.0566	0.1697	1

Table 4.2: Normalized Data set after applying min-max normalization for Tomcat-Class Dataset

wmc	dit	noc	cbo	rfc	lcom	ca
0.0317	0	0	0.0550	0.0274	0.0009	0.0183
0.0079	0.6	0	0.0367	0.0098	0.0000	0.0092
0.2222	0	0	0.4495	0.1742	0.0448	0.3670
0.0159	0.8	0.0323	0	0.0157	0.0002	0
0.0119	0.2	0	0.0275	0.0157	0.0001	0.0275
0.0635	0	0.0645	0.3486	0.1781	0.0020	0.2110
0.0119	0	0	0	0.0059	0.0001	0

ce	npm	lcom3	loc	dam	moa	mfa
0	0.0303	0.5	0.0049	1	0.0833	0
0	0.0087	1	0.0045	0	0	0.9756
0	0.2424	0.4782	0.0733	1	1	0
0	0.0173	1	0.0025	0	0	1
0	0.0130	0.75	0.0092	0	0	0.9375
0	0.0563	0.3889	0.0901	0.8889	0.2917	0
0	0.0130	1	0.0004	0	0	0

cam	ic	cbm	amc	max_cc	avg_cc	bug
0.3929	0	0	0.0039	0.0105	0.075	0
0.6667	0.25	0.0526	0.0190	0.0105	0.05	0
0.1375	0	0	0.0099	0.0526	0.1196	0
0.6667	0	0	0.0045	0	0	0
0.8333	0	0	0.0246	0.0316	0.1	0
0.28	0	0	0.0484	0.1579	0.1938	1
1	0	0	0	0.0105	0.1	0

4.5 Datasets Description

In this work, we have used two datasets that each consisted of 20 attributes and a class label namely 'bug' that is used for classification. Table 4.3 gives the information about dataset descriptions.

Table 4.3: Details of Datasets used for the project

Sr.No.	Dataset Name	Dimensions of Dataset
1	Ant Class	745 observations of 20 attributes
2	Tomcat Class	858 observations of 20 attributes

For the above-preprocessed datasets, first, we applied min-max normalization technique and then we applied ten different feature selection methods for these datasets. Then for each selection method, we derived datasets from the actual datasets with their respective selective attributes. Then, these datasets are fit into the model for classification and cost evaluation for generating 11 results for every dataset, so 22 in overall.

In this study, two open-source software projects are considered to strengthen and generalize our conclusions. In these datasets, object-oriented (OO) metrics related to coupling (CBO, CBM, CA, CE, and IC), cohesion (LCOM, LCOM3, and CAM), inheritance (DIT, NOC, and MFA), encapsulation (DAM and MOA), complexity (WMC, Avg_CC, Max_CC, and AMC), size metrics (LOC and NPM), and RFC measures have been reported, which we have used for performing the experimentations. We fetch the experimental dataset for our work from promise repository 5. Several studies have investigated the capabilities of OO metrics for software fault prediction previously and found that OO metrics performed significantly better in predicting software faults compared to static code metric. Thus, these metrics can be used for building fault prediction models.

Table 4.4 describes the details of the dataset attributes used in the project. Here, these attributes signify the software object-oriented metrics used in the evaluation of various java-based project classes that are faulty or non-faulty in nature based on the fault-based calculated cost.

Table 4.4: Details of dataset attributes (i.e)Software OO metrics used in the project

Sr. No.	Name of the Object-Oriented metric	Description
1	Weighted methods per class(WMC)	Sum of the complexities of methods defined in class
2	Depth of inheritance tree(DIT)	Maximum height of the class hierarchy
3	Number of children(NOC)	Number of immediate descendants of the class
4	Coupling between object classes(CBO)	Number of classes coupled to a given class
5	Response for a Class(RFC)	Number of different methods that can be executed when an object of that class receives a message
6	Lack of cohesion in methods(LCOM)	Number of sets of methods in a class that are not related through the sharing of some of the class's fields
7	Afferent coupling(Ca)	Number of other classes use the specific class
8	Efferent coupling(Ce)	Number of classes used by the specific class
9	Number of public methods(NPM)	Number of methods in a class that are declared as public
10	Lack of cohesion in methods(LCOM3)	Lack of cohesion in methods Henderson-Sellers version
11	Lines of code(LOC)	Number of lines in the text of the source code
12	Data access metric (DAM)	Ratio of the number of private (protected) attributes to the total number of attributes declared in the class
13	Measure of aggregation (MOA)	Number of data declarations (class fields) whose types are user defined classes
14	Measure of functional abstraction (MFA)	Ratio of the number of methods inherited by a class to the total number of methods accessible by member methods of the class
15	Cohesion among methods of class (CAM)	Sum of number of different types of method parameters in every method divided by a multiplication of number of different method parameter types in whole class and number of methods
16	Inheritance coupling (IC)	Number of parent classes to which a given class is coupled
17	Coupling between methods (CBM)	Number of new/redefined methods to which all the inherited methods are coupled
18	Average method complexity (AMC)	Average method size for each class
19	Maximum McCabe's cyclomatic complexity(Max−CC)	Maximum cyclomatic complexity of methods defined in a class
20	Average McCabe's cyclomatic complexity(Avg−CC)	Average cyclomatic complexity of methods defined in a class

CHAPTER 5

INTRODUCTION TO LS-SVM

INTRODUCTION TO LS-SVM

5.1 Implementation:

After carefully examining the dataset it should be pre-processed(which includes Exploratory Analysis and Feature selection methods). The further implementation of LS-SVM takes the following steps

Step 1: Apply the Min-Max Normalization technique to numerical attributes of the dataset.

Step 2: Sample the normalized dataset into training and testing sets for training and prediction. Step 3: Fitting of the datasets into the Least-Square Support Vector Machine model for the three kernel methods.

Step 4: Evaluation of the model using the training dataset.

Step 5: Validation of the model using the testing dataset.

Step 6: Prediction of the model using test data.

Step 7: Generating metrics from the prediction data of the model (i.e.) accuracy,f1-score and confusion matrix

Step 8: Generating the model for cost analysis using the confusion matrix data.

Step 9: Classification of the data items based on the cost model based on its normalized cost

5.2 LS-SVM METHOD:

In this work, Least Square Support Vector Machine (LSSVM) with various kernels have been considered as a classifier for developing a model to classify faulty and non-faulty classes.

The general form of LSSVM function is defined as:

$$y(x) = w^{T} \cdot \Phi(x) + c$$

where , y and x are the output and input vector. $\varphi(x)$ is nonlinear mapping function and it is used for mapping the input data into higher dimensional feature space. w , and c show the adjust weight vector and scalar threshold value respectively.

The objective in LSSVM is to optimize the following equation:

$$\text{Minimize } \frac{1}{2} w^T w + \gamma \frac{1}{2} \sum_{(i=1)}^{1} E_i^2$$

$$\text{Subject w.r.t } y(x) = w^T \Phi(x) + c + E_i, i = 1,2,3,\ldots n$$

where E_i is the error value of input instance *i* and γ is the cost function.

After solving the above optimization problem, fault prediction values are obtained using the below equation:

$$Y' = \sum_{(i=1)}^{1} (\alpha_i - \alpha_i^*) \Phi(x_i) * \Phi(x) + b = \sum_{(i=1)}^{1} (\alpha_i - \alpha_i^*) * K(x_i, x) + b$$

.

where $K(x_i, x)$ is the kernel function, which enables the dot product to be performed in high-dimensional feature space using low- dimensional space data.

The different Kernel functions considered in this study are mentioned below:

1. Linear function
2. Polynomial function
3. Radial basis function

5.2.1 Linear Kernel:

This kernel is used when the data is Linearly separable, that is, it can be separated using a single line. It is mostly used when there are a Large number of Features in a particular Data Set.

It is given by the inner product <x,y> plus an optional constant c.

$$K(x_i, x_j) = x_i^T x_j$$

5.2.2 Polynomial Kernel:

This kernel is non-stationary in nature. Polynomial kernels are well suited for problems where all the training data is normalized.

It is given by the inner product $< X_i , X_j >$ plus an optional constant c ,increasing the power or degree 'd' of the kernel. .

$$K(x_i, x_j) = (x_i^T * x_j + C)^d$$

5.2.3 Gaussian's Radial Basis Function(RBF) Kernel:

This kernel is a function whose value depends on the distance from the origin or from some point.

It is given by the inner product of Euclidean distance between X_i & X_j with parameter γ which is the cost function with an increasing the exponent power of the kernel.

$$K(x_i, x_j) = e^{-\gamma \| x_i - x_j \|^2} \quad , \gamma > 0$$

Value of Gamma(γ) is given by: $\gamma = \frac{1}{2\sigma^2}$ where σ is a scaling constant

5.2.4 Implementation of LSSVM in Python:

```
class LSSVM:
    def __init__(self, gamma=1, kernel='rbf', **kernel_params):
        self.gamma = gamma
        self.x = None
        self.y = None
        self.y_labels = None
        # model params
        self.alpha = None
        self.b = None
        self.kernel = LSSVM.get_kernel(kernel, **kernel_params)
```

```
@staticmethod
def get_kernel(name, **params):
    def linear(x_i, x_j):
        return dot(x_i, x_j.T)
    def poly(x_i, x_j, d=params.get('d',3)):
        return (dot(x_i, x_j . T) + 1) ** d
    def rbf(x_i, x_j, sigma=params.get('sigma', 1)):
        if x_i . ndim == x_i . ndim and x_i . ndim == 2: # both matrices
            return exp(-cdist(x_i, x_j) ** 2 / sigma ** 2)
        else: # both vectors or a vector and a matrix
            return exp(-(dot(x_i, x_i . T) + dot(x_j, x_j . T) -
                    2 *dot(x_i , x_j)) / sigma ** 2)
    kernels = {'linear': linear, 'poly': poly, 'rbf': rbf}
    if kernels.get(name) is None:
        raise KeyError("Kernel '{}' is not defined, try one in the list:
                    {}.".format(name, list(kernels.keys())))
    else:
        return kernels[name]
```

5.2.5 Evaluation and Validation:

Initially, we consider the normalized dataset and then apply train_test_split method in python as the input for the method is the NumPy array format of the dataset split into features and labels. Here, features contain the values of the numeric attributes while labels contain values of the class attribute. In this method, we split the data in the ratio of 80:20 with the records being shuffled. Now, the training set is again split into two for evaluation and validation. We consider the divided training sample and apply the model to perform the evaluation. For the results evaluated, 10-Fold Cross-validation along with fitting of the divided validating sample into the model are done to perform validation. We obtain accuracy in order to compare the results from the evaluation with validation.

5.2.6 Prediction:

We consider the divided testing sample and apply the model to perform the prediction of output values. The predicted values in testing will be compared with actual values to calculate the error by obtaining accuracy for the comparison. Through this result, a confusion matrix is calculated in order to simulate the cost model to find out the costs due to faulty classes and classify the dataset(i.e. Open-source java project) based on its normalized cost.

5.3 COST MODEL ANALYSIS:

This section describes the construction of a cost evaluation model, which accounts for the realistic cost required to remove a fault and compute the estimated fault removal cost for a specific fault prediction technique. Normalized fault removal cost approach suggested by Wagner, has been applied to formulate the proposed cost evaluation model. Cost varies because of various projects being developed on different platforms following several organization standards.

The normalized fault removal costs considered are minimum and are given as:

Table 5.1: Removal costs of test techniques (in staff hour per defects).

Type of test	Costs coefficients
Unit (C_u)	1.5
Integration (C_i)	3.06
System (C_s)	2.82
Field (C_f)	3.9

The fault identification efficiencies for different testing phases considered are maximum and are summarized as :

Table 5.2: Fault identification efficiencies of different test phase. .

Type of test	Costs coefficients
Unit (δ_u)	0.5
Integration (δ_i)	0.6
System (δ_s)	0.65

The formulation of E_{cost} , T_{cost} and the NE_{cost} of the proposed cost based evaluation framework is given as:

5.3.1 Estimated fault removal cost (*E cost*):

$$Cost_{unit} = (No._{F \to F} + No._{NF \to F}) * C_u$$
$$Cost_{integration} = \delta_i * C_i * (No._{F \to NF} + (1 - \delta_u) * No._{F \to F})$$
$$Cost_{system} = \delta_s * C_s * [(1 - \delta_i) * (No._{F \to NF} + (1 - \delta_u) * (No._{F \to F}))]$$
$$Cost_{field} = (1 - \delta_s) * C_f * [(1 - \delta_i) * (No._{F \to NF} + (1 - \delta_u) * No._{F \to F})]$$

$$E_{cost} = C_{initial} + Cost_{unit} + Cost_{integration} + Cost_{system} + Cost_{field}$$

Where $C_{initial} = 0$ for initial setup cost

5.3.2 Estimated testing cost (*T cost*):

$$Cost_{unit} = M_p * C_u * No._{classes}$$
$$Cost_{integration} = \delta_i * C_i * (1 - \delta_u) * No._{faulty\ classes}$$
$$Cost_{system} = \delta_s * C_s * [(1 - \delta_i) * (1 - \delta_u) * No._{faulty\ classes}]$$
$$Cost_{field} = (1 - \delta_s) * C_f * [(1 - \delta_i) * (1 - \delta_u) * No._{faulty\ classes}]$$

$$T_{cost} = C_{initial} + Cost_{unit} + Cost_{integration} + Cost_{system} + Cost_{field}$$

5.3.3 Normalized estimate cost (*NE cost*):

$$NE_{cost} = \frac{E_{cost}}{T_{cost}} \to < 1, \text{ Fault prediction is useful}$$
$$\to \geq 1, \text{ Perform testing}$$

Notations used in the cost evaluation analysis:

1. $C_{initial}$: Initial setup cost for fault-prediction technique applied,
2. C_u , C_i , C_s and C_f are the Normalized fault removal cost in unit, integration, system, and field testing respectively
3. M_p : percentage of classes unit tested.
4. $No._{f \to f}$: number of classes correctly classified as faulty classes

   $No._{nf \to f}$: number of not-faulty classes incorrectly labeled as faulty classes

   $No._{nf \to nf}$: number of not-faulty classes correctly classified as not-faulty classes

   $No._{f \to nf}$: number of faulty classes incorrectly classified as not-faulty classes

   $No._{classes}$: total number of classes

   $No._{faulty\ classes}$: total number of faulty classes
5. δ_u , δ_i , and δ_s are the Fault identification efficiency of unit, integration, and system testing respectively.

5.4 PERFORMANCE EVALUATION PARAMETERS:

All these parameters are computed using a confusion matrix. Confusion matrix contains actual and predicted classifications information done by fault prediction technique.

A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class.

Class	Non faulty	Faulty
Non faulty	$No._{NF \to NF}$	$No._{NF \to F}$
Faulty	$No._{F \to NF}$	$No._{F \to F}$

Confusion matrix to classify a class as faulty and non-faulty

In this study, two performance parameters such as Accuracy and F-Measure are used for measuring the performance of fault prediction models.

Classification accuracy is the total number of correct predictions divided by the total number of predictions made for a dataset.

$$\text{Accuracy} = \frac{\text{No.}_{\text{NF}\rightarrow\text{NF}} + \text{No.}_{\text{F}\rightarrow\text{F}}}{\text{No.}_{\text{NF}\rightarrow\text{NF}} + \text{No.}_{\text{NF}\rightarrow\text{F}} + \text{No.}_{\text{F}\rightarrow\text{NF}} + \text{No.}_{\text{F}\rightarrow\text{F}}}$$

F-Measure or F1-Score provides a single score that balances both the concerns of precision and recall in one number.

Precision quantifies the number of positive class predictions that actually belong to the positive class.

$$\text{Precision} = \frac{\text{No.}_{\text{NF}\rightarrow\text{NF}}}{\text{No.}_{\text{NF}\rightarrow\text{NF}} + \text{No.}_{\text{NF}\rightarrow\text{F}}}$$

Recall quantifies the number of positive class predictions made out of all positive examples in the dataset.

$$\text{Recall} = \frac{\text{No.}_{\text{NF}\rightarrow\text{NF}}}{\text{No.}_{\text{NF}\rightarrow\text{NF}} + \text{No.}_{\text{F}\rightarrow\text{NF}}}$$

The F1 score is the harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

$$\text{F1-Score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$= \frac{\text{No.}_{\text{NF}\rightarrow\text{NF}} + \text{No.}_{\text{F}\rightarrow\text{F}}}{\text{No.}_{\text{NF}\rightarrow\text{NF}} + \text{No.}_{\text{NF}\rightarrow\text{F}} + \text{No.}_{\text{F}\rightarrow\text{NF}} + \text{No.}_{\text{F}\rightarrow\text{F}}}$$

5.5 ARCHITECTURAL METHODOLOGY

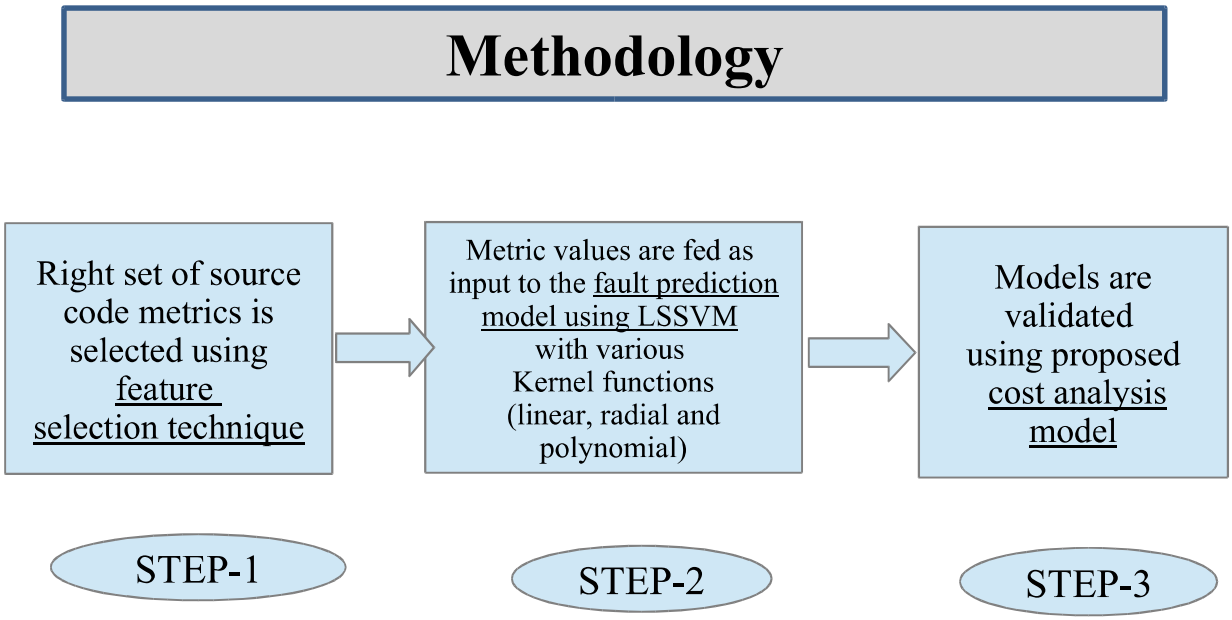

Figure 5.1: Methodology of implemented work

This section describes the architectural methodology implemented in the development of the model, which accounts for three steps. Step 1 consists of the selection of the right set of source code metrics using various feature selection methods which comprise of 4 feature subset selection methods and 6 feature ranking methods. These methods are applied to two datasets which consist of 20 numerical attributes along with a class label. Step 2 involves the values of these selected attributes are given as input to the developed model which is based on the fault prediction functionality using LS-SVM. In this work, we have considered three primary kernel functions namely linear, polynomial and radial basis function. Step 3 involves the validation of results using the proposed cost model analysis which is based on the number of faulty classes that each dataset consists. Normalized fault removal cost approach suggested by Wagner, has been applied to formulate the proposed cost evaluation model. Cost varies because of various projects being developed on different platforms following several organization standards.

5.6 EXPERIMENTAL FRAMEWORK

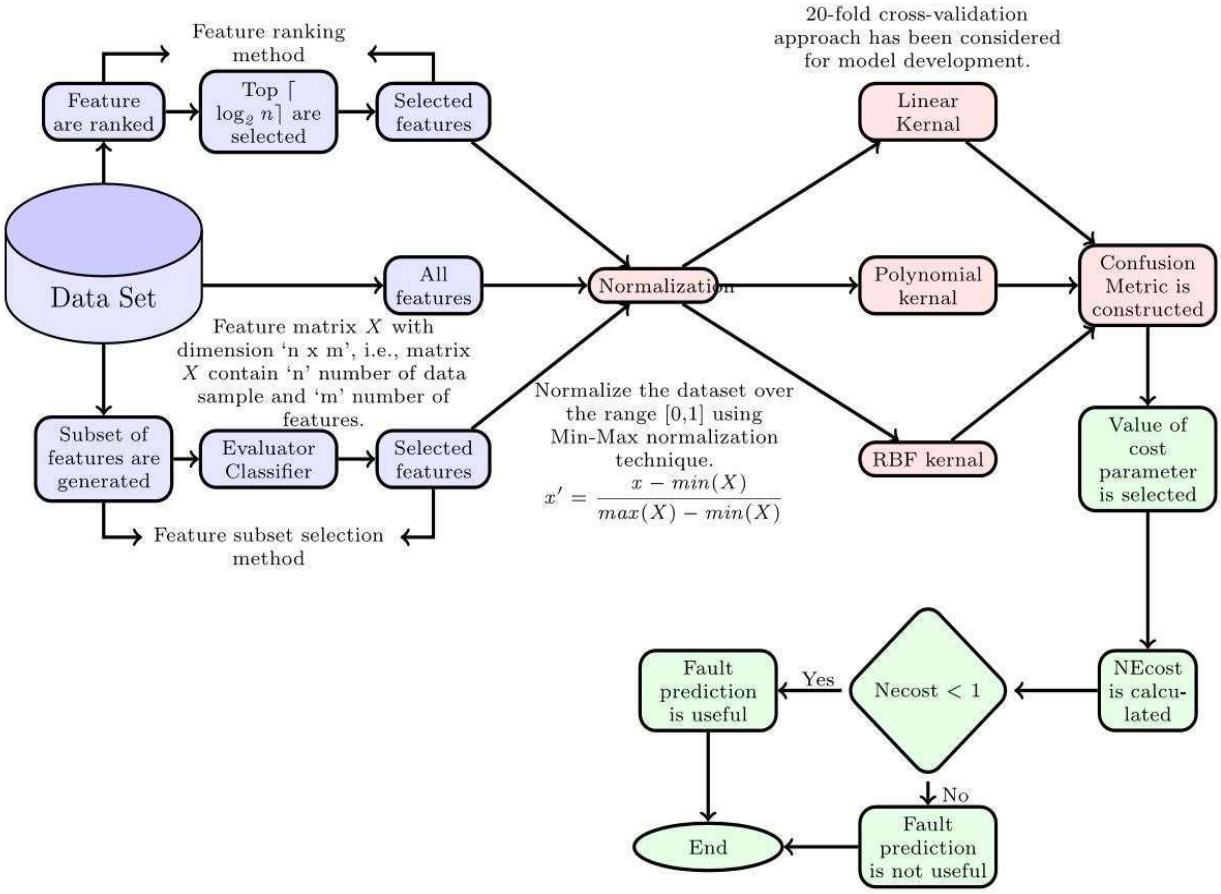

Figure 5.2: Experimental Framework of the project

This section describes the experimental framework of the developed work. Initially, from the preprocessed datasets various features are obtained from feature selection methods which comprise of 4 feature subset selection methods and 6 feature ranking methods. Even the original dataset along with the feature selection methods is considered for processing of model. Then the values of these selected features and all features are normalized using Min-Max normalization in the range of [0,1]. These normalized values of selected attributes are given as input to the developed model which is based on the fault prediction functionality using LS-SVM. In this work, we have considered three primary kernel functions namely linear, polynomial and radial basis function. A confusion matrix is constructed from the fault prediction model. The validation of results using the proposed cost model analysis which is based on the number of faulty classes that each dataset consists of selecting value of cost parameter. For the normalized cost calculated, if the normalized cost is greater than 1, it is regarded as a useful fault prediction. Else, the fault prediction is regarded as not useful.

CHAPTER 7
RESULTS AND DISCUSSION

RESULTS AND DISCUSSION

7.1 Comparisons

7.1.1 Feature selection techniques applied to the given datasets:

The below tables describe the list of attributes selected for the two datasets using 4 feature subset selection methods, 6 feature ranking methods and also including the sample dataset . In the case of sample dataset, all the attributes are considered in the attribute selection. In the case of feature ranking methods, using ranker methods the attributes are ranked based on the values of their coefficients and only top 5 attributes are considered in the selected attributes. In the case of feature subset selection methods, attributes are selected based on the backward greedy stepwise method and the selected attributes may be in any number.

Table 7.1.1 describes the list of attributes selected for the Ant class dataset using feature subset selection methods and feature ranking methods.

Table 7.1.1: List of attributes selected for Ant dataset using feature subset and ranking methods

Technique Used	Attributes Selected(ie. Used for Ant dataset)
Sample Data Set	All attributes selected
Chi-Squared Test	rfc,loc,max_cc,ce,wmc
Gain Ratio	rfc,loc,cam,ce,max_cc
Information Gain	rfc,loc,wmc,ce,lcom
One-R Evaluation	ca,dam,cbo,noc,ic
Logistic Regression Analysis	wmc,amc,loc,rfc,lcom
Principal Component Analysis	wmc,dit,noc,cbo,rfc
Correlation Based Feature Selection	rfc,lcom,ce,loc,moa,cam,amc,max_cc
Rough Set Analysis	dit,cbo,rfc,lcom,ca,ce,lcom3,loc,dam,moa,ic,amc, avg_cc
Consistency Subset Evaluation	wmc,cbo,rfc,lcom,ce,npm,lcom3,loc,moa,cam,amc, max_cc,avg_cc
Filtered Subset Evaluation	rfc,lcom,ce,loc,moa,cam,amc,max_cc

Table 7.1.2 describes the list of attributes selected for the Tomcat class dataset using feature subset selection methods and feature ranking methods.

Table 7.1.2: List of attributes selected for Tomcat dataset using feature subset and ranking methods

Technique Used	Attributes Selected(ie. Used for Tomcat dataset)
Sample Data Set	All attributes selected
Chi-Squared Test	rfc,loc,max_cc,lcom3,moa
Gain Ratio	rfc,loc,max_cc,moa,npm
Information Gain	rfc,loc,cbo,max_cc,lcom3
One-R Evaluation	rfc,loc,dit,noc,ce
Logistic Regression Analysis	rfc,npm,ca,moa,wmc
Principal Component Analysis	wmc,dit,noc,cbo,rfc
Correlation Based Feature Selection	cbo,rfc,lcom,npm,lcom3,loc,moa,amc,max_cc,avg_cc
Rough Set Analysis	wmc,dit,cbo,lcom,ca,npm,lcom3,loc,cam,amc,max_cc,avg_cc
Consistency Subset Evaluation	wmc,rfc,lcom,ca,npm,lcom3,loc,dam,moa,cam,amc,max_cc,avg_cc
Filtered Subset Evaluation	cbo,rfc,lcom,npm,lcom3,loc,moa,amc,max_cc,avg_cc

These numerical attributes of various methods are now given as input to the fault prediction model and thereby generating results of Evaluation, Validation and Prediction. The below sections describe the results of evaluation, validation and prediction of the two datasets across the three kernels. The comparison metric used in the result are Accuracy Score , Cross-Validation Score and F1-Score.

7.1.2 Evaluation results of LSSVM model for the two dataset :

Evaluation is a systematic determination of a subject's merit, worth and significance, using criteria governed by a set of standards. This section deals with the illustration of the evaluation results of the fault prediction model for the two datasets across the three kernels. Accuracy, Cross-Validation Score and F1-Score are the metrics used for the comparison between the actual dataset and the ten feature selection methods.

a. For Linear Kernel

Table 7.2.1: Evaluation of Accuracy,F1-Score and Cross-validation score for Linear Kernel

Dataset Name	Linear Kernel_Acc	Linear Kernel_F1	Linear Kernel_CV
Ant_AM	78.151	0.888	85.7
Ant_CS	77.3109	0.876	86.1303
Ant_GR	77.1	0.857	84.033
Ant_IG	77.731	0.873	86.551
Ant_LR	76.68	0.8571	83
Ant_OneR	75.63	0.8655	86.35
Ant_PCA	78.571	0.8823	85.7
Ant_CFS	77.3109	0.8697	84.8759
Ant_CSE	79.2016	0.8802	86.3475
Ant_FSE	75.4201	0.8676	84.8803
Ant_RSA	76.89	0.8655	84.459
Tomcat_AM	91.605	0.945	94.35
Tomcat_CS	90.51	0.93	92.329
Tomcat_GR	91.605	0.941	94.168
Tomcat_IG	92.153	0.943	94.333
Tomcat_LR	91.24	0.937	93.42
Tomcat_OneR	91.9708	0.939	93.973
Tomcat_PCA	89.963	0.925	91.791
Tomcat_CFS	89.781	0.921	91.606
Tomcat_CSE	92.153	0.939	93.801
Tomcat_FSE	92.144	0.925	89.963
Tomcat_RSA	90.693	0.9306	91.79

Highest accuracy in Ant dataset is showed in the case of Consistency-Subset Evaluation(79.20) whereas the least accuracy is showed in the case of its actual dataset(73.31). As of Tomcat dataset, accuracy was high in the case of Consistency-Subset Evaluation and Information Gain(92.153) whereas the least accuracy is shown in the case of CFS Subset Evaluation(89.781). F1-Score of Ant dataset varies in the range of [0.857, 0.886] while that of Tomcat dataset varies in the range of [0.925,0.945].

b. For Polynomial Kernel

Table 7.2.2: Evaluation of Accuracy,F1-Score and Cross-validation score for Polynomial Kernel

Dataset Name	Poly Kernel_Acc	Poly Kernel_F1	Poly Kernel_CV
Ant_AM	73.319	0.886	80.234
Ant_CS	77.3109	0.876	86.546
Ant_GR	77.1	0.857	83.648
Ant_IG	77.731	0.8739	86.1347
Ant_LR	76.68	0.857	82.567
Ant_OneR	75.63	0.865	85.496
Ant_PCA	78.5714	0.8823	87.1852
Ant_CFS	77.3109	0.8697	84.6631
Ant_CSE	79.2016	0.8802	83.4219
Ant_FSE	75.4201	0.8676	84.0425
Ant_RSA	76.89	0.8655	80.855
Tomcat_AM	91.605	0.945	90.148
Tomcat_CS	90.51	0.93	92.87
Tomcat_GR	91.605	0.941	93.969
Tomcat_IG	92.153	0.943	94.34
Tomcat_LR	91.24	0.937	93.26
Tomcat_OneR	91.97	0.939	93.98
Tomcat_PCA	89.963	0.925	90.878
Tomcat_CFS	89.781	0.921	89.235
Tomcat_CSE	92.153	0.939	90.875
Tomcat_FSE	89.963	0.9251	89.043
Tomcat_RSA	90.693	0.9306	91.404

Highest accuracy in Ant dataset is showed in the case of Consistency-Subset Evaluation(79.20) whereas the least accuracy is showed in the case of its actual dataset(73.31). As of Tomcat dataset, accuracy was high in the case of Consistency-Subset Evaluation and Information Gain(92.153) whereas the least accuracy is shown in the case of CFS Subset Evaluation(89.781). F1-Score of Ant dataset varies in the range of [0.857, 0.886] while that of Tomcat dataset varies in the range of [0.925,0.945]. The metric values (i.e. Accuracy, F1-Score and Cross-Validation score) of Ant dataset is low compared to Tomcat dataset because the number of faulty classes is more in the case of Ant dataset compared to Tomcat dataset. As the number of faulty classes increases, the performance of the model decreases.

c. For RBF Kernel

Table 7.2.3: Evaluation of Accuracy,F1-Score and Cross-validation score for RBF Kernel

Dataset Name	RBF Kernel_Acc	RBF Kernel_F1	RBF Kernel_CV
Ant_AM	78.151	0.888	88.8608
Ant_CS	77.3109	0.876	87.384
Ant_GR	77.1	0.857	85.704
Ant_IG	77.731	0.8739	87.3803
Ant_LR	76.68	0.857	85.735
Ant_OneR	75.63	0.865	85.0664
Ant_PCA	78.571	0.882	87.832
Ant_CFS	77.3109	0.8697	86.9592
Ant_CSE	79.2016	0.8802	88.027
Ant_FSE	75.4201	0.8676	86.777
Ant_RSA	76.89	0.8655	86.555
Tomcat_AM	91.605	0.945	94.521
Tomcat_CS	90.51	0.93	92.88
Tomcat_GR	91.605	0.941	93.969
Tomcat_IG	92.153	0.943	94.168
Tomcat_LR	91.24	0.937	93.791
Tomcat_OneR	91.97	0.939	93.97
Tomcat_PCA	89.963	0.925	92.525
Tomcat_CFS	89.781	0.921	92.158
Tomcat_CSE	92.153	0.939	93.983
Tomcat_FSE	89.963	0.925	92.518
Tomcat_RSA	90.693	0.9306	93.07

The metric values (i.e. Accuracy, F1-Score and Cross-Validation score) of Ant dataset is low compared to Tomcat dataset because the number of faulty classes is more in the case of Ant dataset compared to Tomcat dataset. As the number of faulty classes increases, the performance of the model decreases. Highest accuracy in Ant dataset is showed in the case of Consistency-Subset Evaluation(79.20) whereas the least accuracy is showed in the case of its actual dataset(73.31). As of Tomcat dataset, accuracy was high in the case of Consistency-Subset Evaluation and Information Gain(92.153) whereas the least accuracy is shown in the case of CFS Subset Evaluation(89.781). F1-Score of Ant dataset varies in the range of [0.857, 0.886] while that of Tomcat dataset varies in the range of [0.925,0.945].

7.1.3 Validation results of LSSVM model for the two dataset :

Validation is the process of establishing documentary evidence demonstrating that a procedure, process, or activity carried out in testing and then production maintains the desired level of compliance at all stages. This section deals with the illustration of the validation results of the fault prediction model for the two datasets across the three kernels. Accuracy, Cross-Validation Score and F1-Score are the metrics used for the comparison between the actual dataset and the ten feature selection methods.

a. For Linear Kernel

Table 7.2.4: Validation of Accuracy,F1-Score and Cross-validation score for Linear Kernel

Dataset Name	Linear Kernel_Acc	Linear Kernel_F1	Linear Kernel_CV
Ant_AM	78.333	0.84166	75.833
Ant_CS	82.5	0.925	88.33
Ant_GR	74.167	0.875	85.833
Ant_IG	75.833	0.891	86.667
Ant_LR	80	0.925	92.5
Ant_OneR	82.5	0.9083	90.833
Ant_PCA	76.667	0.891	85.833
Ant_CFS	75.833	0.84166	85
Ant_CSE	75.833	0.85	79.166
Ant_FSE	78.333	0.875	86.667
Ant_RSA	79.166	0.8833	85.8333
Tomcat_AM	92.028	0.934	90.549
Tomcat_CS	92.028	0.934	92.747
Tomcat_GR	89.855	0.927	93.516
Tomcat_IG	86.231	0.905	87.692
Tomcat_LR	86.956	0.905	88.351
Tomcat_OneR	89.855	0.9202	92.03
Tomcat_PCA	95.652	0.978	97.087
Tomcat_CFS	92.753	0.949	94.285
Tomcat_CSE	89.13	0.934	90.549
Tomcat_FSE	92.753	0.942	91.373
Tomcat_RSA	93.478	0.956	94.175

Highest accuracy in Ant dataset is showed in the case of Chi-Square Evaluation and One R Evaluation(82.5) whereas the least accuracy is showed in the case of Gain Ratio(74.16). As of Tomcat dataset, accuracy was high in the case of Principal Component Analysis(95.652) whereas the least accuracy is shown in the case of Information Gain(86.231). F1-Score of Ant dataset varies in the range of [0.841, 0.925] while that of Tomcat dataset varies in the range of [0.905,0.978].

b. For Polynomial Kernel

Table 7.2.5: Validation of Accuracy,F1-Score and Cross-validation score for Polynomial Kernel

Dataset Name	Poly Kernel_Acc	Poly Kernel_F1	Poly Kernel_CV
Ant_AM	78.333	0.84166	71.666
Ant_CS	82.5	0.925	86.667
Ant_GR	74.167	0.875	80.833
Ant_IG	75.833	0.891	85
Ant_LR	80	0.925	90.833
Ant_OneR	82.5	0.908	90.833
Ant_PCA	76.667	0.891	85.833
Ant_CFS	75.833	0.84166	77.5
Ant_CSE	75.833	0.85	63.33
Ant_FSE	78.333	0.8667	76.667
Ant_RSA	79.166	0.8833	71.667
Tomcat_AM	92.028	0.934	75.989
Tomcat_CS	92.028	0.934	91.318
Tomcat_GR	89.855	0.927	90.549
Tomcat_IG	86.231	0.905	85.494
Tomcat_LR	86.956	0.905	87.747
Tomcat_OneR	89.855	0.9202	89.12
Tomcat_PCA	95.652	0.978	97.857
Tomcat_CFS	92.753	0.9492	83.461
Tomcat_CSE	89.13	0.934	83.351
Tomcat_FSE	92.753	0.942	79.01
Tomcat_RSA	93.478	0.9565	89.01

Highest accuracy in Ant dataset is showed in the case of Chi-Square Evaluation and One R Evaluation(82.5) whereas the least accuracy is showed in the case of Gain Ratio(74.16). As of Tomcat dataset, accuracy was high in the case of Principal Component Analysis(95.652) whereas the least accuracy is shown in the case of Information Gain(86.231). F1-Score of Ant dataset varies in the range of [0.841, 0.925] while that of Tomcat dataset varies in the range of [0.905,0.978].The metric values (i.e. Accuracy, F1-Score and Cross-Validation score) of Ant dataset is low compared to Tomcat dataset because the number of faulty classes is more in the case of Ant dataset compared to Tomcat dataset. As the number of faulty classes increases, the performance of the model decreases.

c. For RBF Kernel

Table 7.2.6: Validation of Accuracy,F1-Score and Cross-validation score for RBF Kernel

Dataset Name	RBF Kernel_Acc	RBF Kernel_F1	RBF Kernel_CV
Ant_AM	78.333	0.84166	84.1666
Ant_CS	82.5	0.925	92.5
Ant_GR	74.167	0.875	86.667
Ant_IG	75.833	0.891	89.1667
Ant_LR	80	0.925	92.5
Ant_OneR	82.5	0.90833	89.1667
Ant_PCA	76.667	0.891	89.1667
Ant_CFS	75.833	0.84166	84.1666
Ant_CSE	75.833	0.85	85
Ant_FSE	78.333	0.875	87.5
Ant_RSA	79.166	0.8833	88.33
Tomcat_AM	92.028	0.934	93.4615
Tomcat_CS	92.028	0.934	93.516
Tomcat_GR	89.855	0.927	92.747
Tomcat_IG	86.231	0.905	90.659
Tomcat_LR	86.956	0.905	90.659
Tomcat_OneR	89.855	0.9202	91.978
Tomcat_PCA	95.652	0.978	97.747
Tomcat_CFS	92.753	0.949	94.945
Tomcat_CSE	89.13	0.934	93.4
Tomcat_FSE	92.753	0.942	94.175
Tomcat_RSA	93.478	0.9565	95.549

The metric values (i.e. Accuracy, F1-Score and Cross-Validation score) of Ant dataset is low compared to Tomcat dataset because the number of faulty classes is more in the case of Ant dataset compared to Tomcat dataset. As the number of faulty classes increases, the performance of the model decreases. Highest accuracy in Ant dataset is showed in the case of Chi-Square Evaluation and One R Evaluation(82.5) whereas the least accuracy is showed in the case of Gain Ratio(74.16). As of Tomcat dataset, accuracy was high in the case of Principal Component Analysis(95.652) whereas the least accuracy is shown in the case of Information Gain(86.231). F1-Score of Ant dataset varies in the range of [0.841, 0.925] while that of Tomcat dataset varies in the range of [0.905,0.978].

7.1.4 Prediction results of LSSVM model for the two dataset :

Prediction refers to the output of an algorithm after it has been trained on a historical dataset and applied to new data when forecasting the likelihood of a particular outcome This section deals with the illustration of the prediction results of the fault prediction model for the two datasets across the three kernels. Accuracy, Cross-Validation Score and F1-Score are the metrics used for the comparison between the actual dataset and the ten feature selection methods.

a. For Linear Kernel

Table 7.2.7: Prediction of Accuracy,F1-Score and Cross-validation score for Linear Kernel

Dataset Name	Linear Kernel_Acc	Linear Kernel_F1	Linear Kernel_CV
Ant_AM	75.838	0.859	79.9047
Ant_CS	75.167	0.8322	83.33
Ant_GR	82.55	0.932	91.33
Ant_IG	79.1946	0.8657	82.4285
Ant_LR	79.1946	0.892	89.33
Ant_OneR	80.536	0.8791	87.904
Ant_PCA	75.8389	0.8389	81.1428
Ant_CFS	80.536	0.919	91.952
Ant_CSE	74.496	0.8791	84.3809
Ant_FSE	84.5637	0.8993	87.2857
Ant_RSA	79.1946	0.8993	87.1904
Tomcat_AM	88.372	0.9069	87.78
Tomcat_CS	91.86	0.953	95.915
Tomcat_GR	90.116	0.924	93.039
Tomcat_IG	91.279	0.936	94.215
Tomcat_LR	93.604	0.953	93.529
Tomcat_OneR	88.953	0.936	93.6601
Tomcat_PCA	90.697	0.936	93.627
Tomcat_CFS	93.604	0.9709	97.156
Tomcat_CSE	88.953	0.924	91.209
Tomcat_FSE	93.02	0.965	96.535
Tomcat_RSA	90.116	0.936	92.483

Highest accuracy in Ant dataset is showed in the case of Filtered Subset Evaluation(84.56) whereas the least accuracy is showed in the case of Consistency-Subset Evaluation(74.49). As of Tomcat dataset, accuracy was high in the case of Consistency-Subset Evaluation and Logistic Regression Analysis(93.604) whereas the least accuracy is shown in the case of its actual dataset(88.372). F1-Score of Ant dataset varies in the range of [0.832, 0.932] while that of Tomcat dataset varies in the range of [0.906,0.971].

b. For Polynomial Kernel

Table 7.2.8: Prediction of Accuracy,F1-Score and Cross-validation score for Polynomial Kernel

Dataset Name	Poly Kernel_Acc	Poly Kernel_F1	Poly Kernel_CV
Ant_AM	75.838	0.859	67.142
Ant_CS	75.167	0.8322	75.8095
Ant_GR	82.55	0.932	87.904
Ant_IG	79.1946	0.8657	83.8095
Ant_LR	79.1946	0.892	85.285
Ant_OneR	80.53	0.8791	83.809
Ant_PCA	75.838	0.838	84.571
Ant_CFS	88.571	0.919	88.5714
Ant_CSE	73.825	0.8322	67.047
Ant_FSE	84.5637	0.8993	85.8095
Ant_RSA	79.1946	0.8993	85.238
Tomcat_AM	87.2	0.895	73.986
Tomcat_CS	91.86	0.953	95.326
Tomcat_GR	90.116	0.924	91.83
Tomcat_IG	91.279	0.936	93.039
Tomcat_LR	93.604	0.953	94.183
Tomcat_OneR	88.95	0.936	91.83
Tomcat_PCA	90.697	0.936	91.862
Tomcat_CFS	93.604	0.9709	93.66
Tomcat_CSE	88.95	0.924	87.745
Tomcat_FSE	93.023	0.965	85.947
Tomcat_RSA	90.116	0.936	85.55

Highest accuracy in Ant dataset is showed in the case of Filtered Subset Evaluation(84.56) whereas the least accuracy is showed in the case of Consistency-Subset Evaluation(73.82). As of Tomcat dataset, accuracy was high in the case of Consistency-Subset Evaluation and Logistic Regression Analysis(93.604) whereas the least accuracy is shown in the case of its actual dataset(87.2). F1-Score of Ant dataset varies in the range of [0.832, 0.932] while that of Tomcat dataset varies in the range of [0.895,0.971]. The metric values (i.e. Accuracy, F1-Score and Cross-Validation score) of Ant dataset is low compared to Tomcat dataset because the number of faulty classes is more in the case of Ant dataset compared to Tomcat dataset. As the number of faulty classes increases, the performance of the model decreases.

c. For RBF Kernel

Table 7.2.9: Prediction of Accuracy,F1-Score and Cross-validation score for RBF Kernel

Dataset Name	RBF Kernel_Acc	RBF Kernel_F1	RBF Kernel_CV
Ant_AM	75.838	0.859	85.952
Ant_CS	75.167	0.8322	83.1904
Ant_GR	82.55	0.932	93.238
Ant_IG	79.1946	0.8657	86.5714
Ant_LR	79.1946	0.892	89.333
Ant_OneR	63.7583	0.7114	87.2857
Ant_PCA	75.838	0.838	83.809
Ant_CFS	80.536	0.919	91.857
Ant_CSE	74.496	0.8791	87.857
Ant_FSE	84.563	0.8993	90
Ant_RSA	79.1946	0.8993	89.904
Tomcat_AM	88.372	0.906	90.816
Tomcat_CS	91.8604	0.9534	95.326
Tomcat_GR	90.116	0.9244	92.418
Tomcat_IG	91.279	0.936	93.692
Tomcat_LR	93.604	0.953	95.457
Tomcat_OneR	88.953	0.936	93.594
Tomcat_PCA	90.697	0.936	93.594
Tomcat_CFS	93.604	0.9709	97.0915
Tomcat_CSE	88.953	0.924	92.45
Tomcat_FSE	93.023	0.9651	96.503
Tomcat_RSA	90.116	0.936	93.627

The metric values (i.e. Accuracy, F1-Score and Cross-Validation score) of Ant dataset is low compared to Tomcat dataset because the number of faulty classes is more in the case of Ant dataset compared to Tomcat dataset. As the number of faulty classes increases, the performance of the model decreases. Highest accuracy in Ant dataset is showed in the case of Filtered Subset Evaluation(84.563) whereas the least accuracy is showed in the case of One R Evaluation(63.75). As of Tomcat dataset, accuracy was high in the case of CFS Subset Evaluation and Logistic Regression Analysis(93.604) whereas the least accuracy is shown in the case of its actual dataset (88.372). F1-Score of Ant dataset varies in the range of [0.711, 0.932] while that of Tomcat dataset varies in the range of [0.906,0.971].

7.1.5 Cost Model Analysis for the given datasets across three kernels of LS-SVM:

The cost required to remove a fault is calculated based on the fault removal costs and the fault identification efficiencies. Unit, integration, system and field testing techniques are applied during the calculation of fault removal cost. Estimated fault removal cost involves the calculation of fault removal cost for software when the fault prediction approach is performed using various testing phases. Estimated testing cost involves the calculation of fault removal cost for software when the fault prediction approach is not performed. Table 7.3.1 shows the values of the costs for removal of faults for various feature selection methods along with the actual dataset across three kernels(linear, polynomial and radial basis function).

The cost depends on three factors. Higher are the number of faulty classes, more is the value of the fault removal costs. Cost varies because of various projects being developed on different platforms following several organization standards because they accept various fault removal cost and fault identification coefficients.

Table 7.3.1: Estimation and Total costs across three kernels of LS-SVM for the datasets

Dataset Name	Estimstion Cost in Linear Kernel	Total Cost in Linear kernel	Estimation Cost in Polynomial kernel	Total Cost in Polynomial kernel	Estimstion Cost in RBF Kernel	Total Cost in RBF kernel
Ant_AM	65.419	32.710	73.238	65.914	73.238	65.914
Ant_CS	77.880	38.940	87.188	78.469	87.188	78.469
Ant_GR	31.152	15.576	34.875	31.387	34.875	31.387
Ant_IG	62.304	31.152	69.750	62.775	69.750	62.775
Ant_LR	49.843	24.922	55.800	50.220	55.800	50.220
Ant_OneR	56.074	28.037	62.775	56.497	108.228	181.466
Ant_PCA	74.765	37.824	83.700	75.330	83.700	75.330
Ant_CFS	37.382	18.691	41.850	37.665	41.850	37.665
Ant_CSE	56.074	28.037	75.926	91.969	62.775	56.497
Ant_FSE	76.728	23.364	52.313	47.081	52.313	47.081
Ant_RSA	46.728	23.364	52.313	47.081	52.313	47.081
Tomcat_AM	49.843	24.922	58.800	59.497	55.800	50.220
Tomcat_CS	24.922	12.461	27.900	25.110	27.900	25.110
Tomcat_GR	40.498	20.249	45.338	40.804	45.338	40.804
Tomcat_IG	34.267	17.134	38.363	34.526	38.363	34.526
Tomcat_LR	24.922	12.461	27.900	25.110	27.900	25.110
Tomcat_OneR	34.267	17.134	38.363	34.526	38.363	34.527
Tomcat_PCA	34.267	17.134	38.363	34.526	38.363	34.527
Tomcat_CFS	15.576	7.788	17.438	15.694	17.438	15.694
Tomcat_CSE	40.498	20.249	45.338	40.804	45.338	40.804
Tomcat_FSE	18.691	9.346	20.925	18.832	20.925	18.832
Tomcat_RSA	34.267	17.134	38.363	34.527	38.363	34.526

Normalized estimated fault removal cost is defined as the ratio of Estimated fault removal cost to the Estimated testing cost of the software when fault prediction is utilized. Table 7.3.2 shows the normalized cost values for removal of faults for various feature selection methods along with the actual dataset across three kernels(linear, polynomial and radial basis function).

If the normalized estimated cost is greater than 1, then the fault prediction is considered not useful. Else, the fault prediction is useful.

Table 7.3.2: Normalized costs across three kernels of LS-SVM for the datasets

Dataset Name	Normalized Cost in Linear kernel	Normalized Cost in Polynomial kernel	Normalized Cost in RBF kernel
Ant_AM	2	1.111	1.111
Ant_CS	1.999	1.111	1.111
Ant_GR	2	1.111	1.111
Ant_IG	2	1.111	1.111
Ant_LR	2	1.111	1.111
Ant_OneR	2	1.111	0.596
Ant_PCA	2	1.111	1.111
Ant_CFS	2	1.111	1.111
Ant_CSE	2	0.8255	1.111
Ant_FSE	2	1.111	1.111
Ant_RSA	2	1.111	1.111
Tomcat_AM	2	0.9882	1.111
Tomcat_CS	2	1.111	1.111
Tomcat_GR	2	1.111	1.111
Tomcat_IG	2	1.111	1.111
Tomcat_LR	2	1.111	1.111
Tomcat_OneR	2	1.111	1.111
Tomcat_PCA	2	1.111	1.111
Tomcat_CFS	2	1.111	1.111
Tomcat_CSE	2	1.111	1.111
Tomcat_FSE	2	1.111	1.111
Tomcat_RSA	2	1.111	1.111

Lower the value of Normalized estimation cost means that the estimated fault removal cost is lower than the estimated testing cost resulting in good fault prediction. Whereas, higher the value of Normalized estimation cost means that the estimated fault removal cost is higher than the estimated testing cost resulting in performing of testing. None of the linear kernels resulted in good fault prediction. While Consistency Subset Evaluation of Ant dataset and actual dataset of Tomcat dataset resulted in good predication in the polynomial kernel. Whereas, OneR Evaluation of Ant dataset resulted in good predication in the RBF kernel.

7.2 Plots

Accuracies of all comparisons across all the three kernels for AntClass dataset

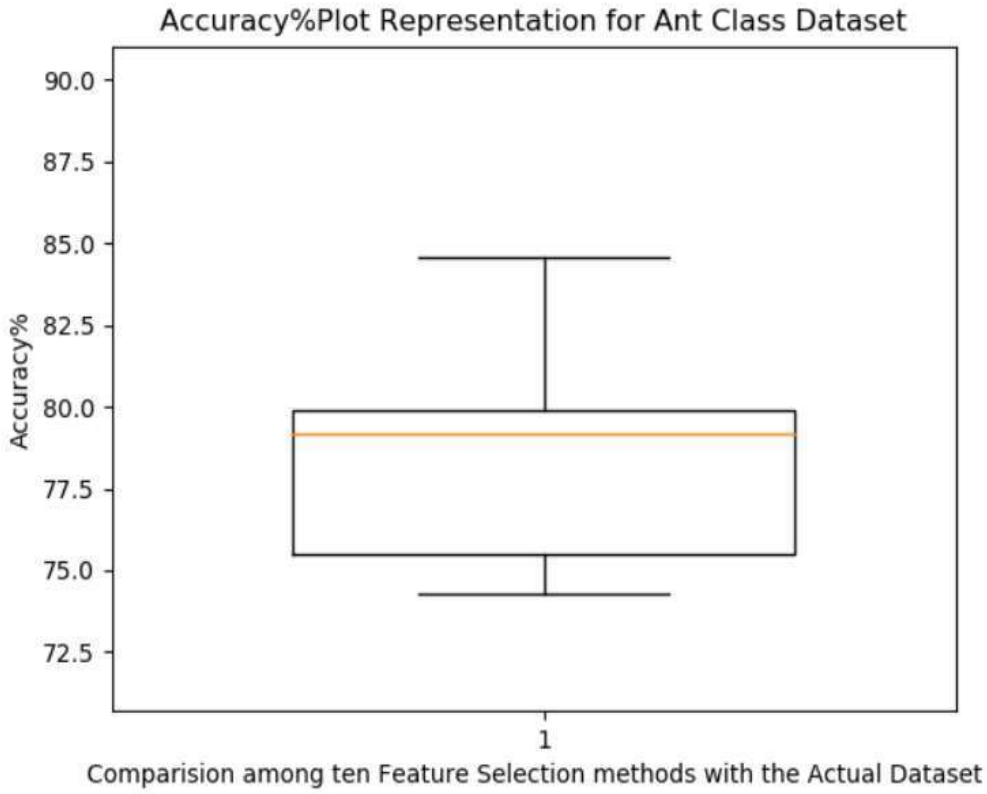

Fig 7.1: Boxplot representation of accuracies for AntClass dataset

Accuracies of all comparisons:

Dataset Name	Accuracies across all kernels
Ant_AM	75
Ant_CS	75
Ant_GR	82
Ant_IG	79
Ant_LR	79
Ant_OneR	74
Ant_PCA	75
Ant_CFS	80
Ant_CSE	74
Ant_FSE	84
Ant_RSA	79

For the comparison of accuracies of all the methods including the actual dataset across all the kernels, the highest accuracy in the Ant dataset is recorded in the case of Filtered Subset Evaluation(84) while the least accuracy is recorded in the case of One R Evaluation and CFS Subset Evaluation(74).

Accuracies of all comparisons across all the three kernels for Tomcat-Class dataset

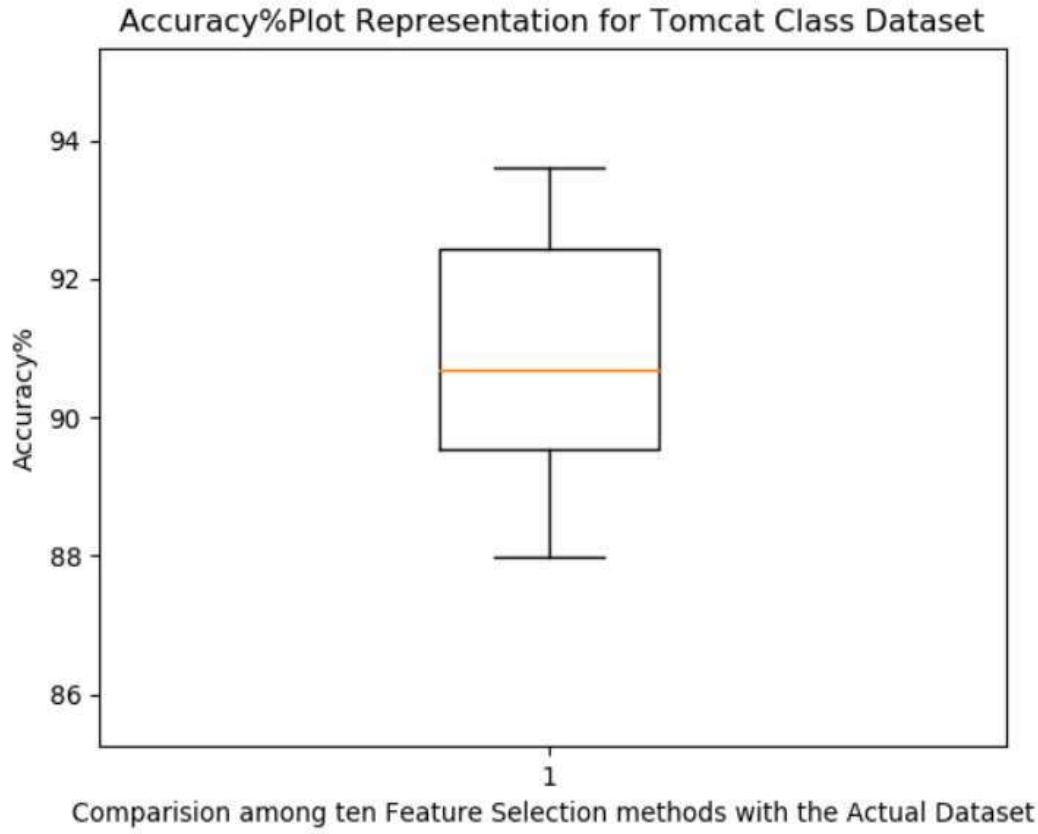

Fig 7.2: Boxplot representation of accuracies for TomcatClass dataset

Accuracies of all comparisons:

Dataset Name	Accuracies across all kernels
Tomcat_AM	87
Tomcat_CS	91
Tomcat_GR	90
Tomcat_IG	91
Tomcat_LR	93
Tomcat_OneR	88
Tomcat_PCA	90
Tomcat_CFS	93
Tomcat_CSE	88
Tomcat_FSE	93
Tomcat_RSA	90

For the comparison of accuracies of all the methods including the actual dataset across all the kernels, the highest accuracy in the Tomcat dataset is recorded in the case of Filtered Subset Evaluation, Logistic Regression and CFS Subset Evaluation(93) while the least accuracy is recorded in the case of its actual dataset(87).

F1-Scores of all comparisons across all the three kernels for Ant-Class dataset

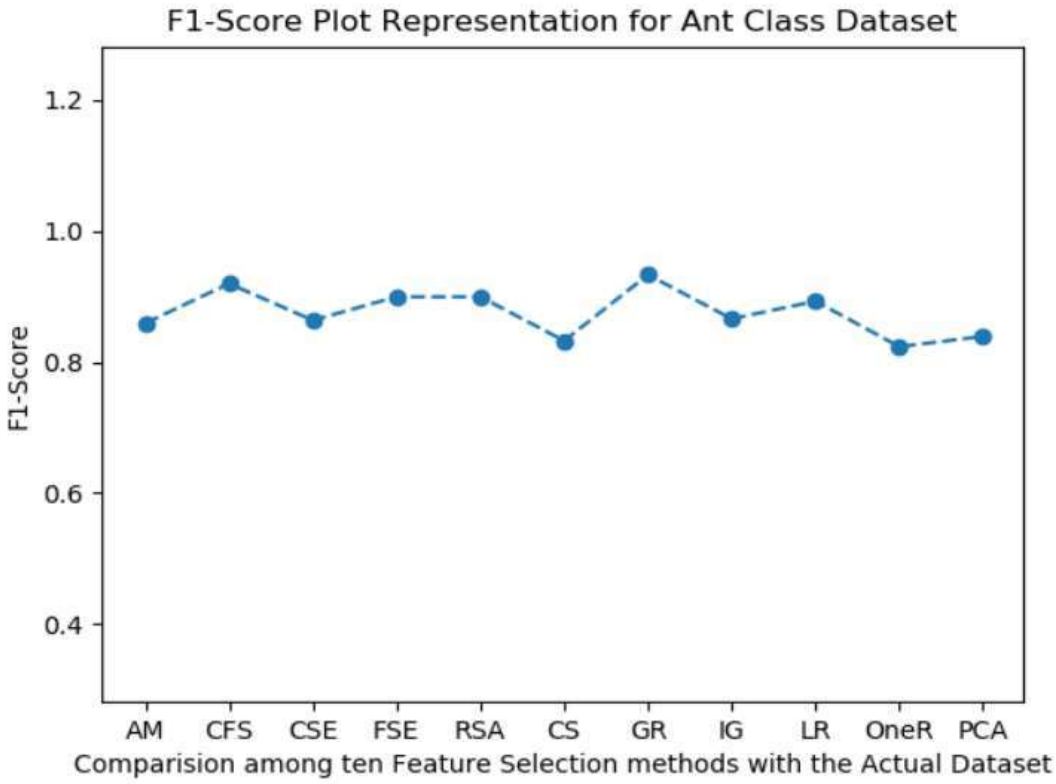

Fig 7.3: Line representation of F1-Scores for AntClass dataset

F1-Scores of all comparisons:

Dataset Name	F1-Scores across all kernels
Ant_AM	0.859
Ant_CS	0.832
Ant_GR	0.932
Ant_IG	0.865
Ant_LR	0.892
Ant_OneR	0.823
Ant_PCA	0.838
Ant_CFS	0.919
Ant_CSE	0.863
Ant_FSE	0.899
Ant_RSA	0.899

For the comparison of F1-Scores of all the methods including the actual dataset across all the kernels, the maximum f1-score in the Ant dataset is recorded in the case of Gain Ratio Evaluation(0.932) while the minimum f1-score is recorded in the case of One R Evaluation(0.823).

F1-Scores of all comparisons across all the three kernels for Tomcat-Class dataset

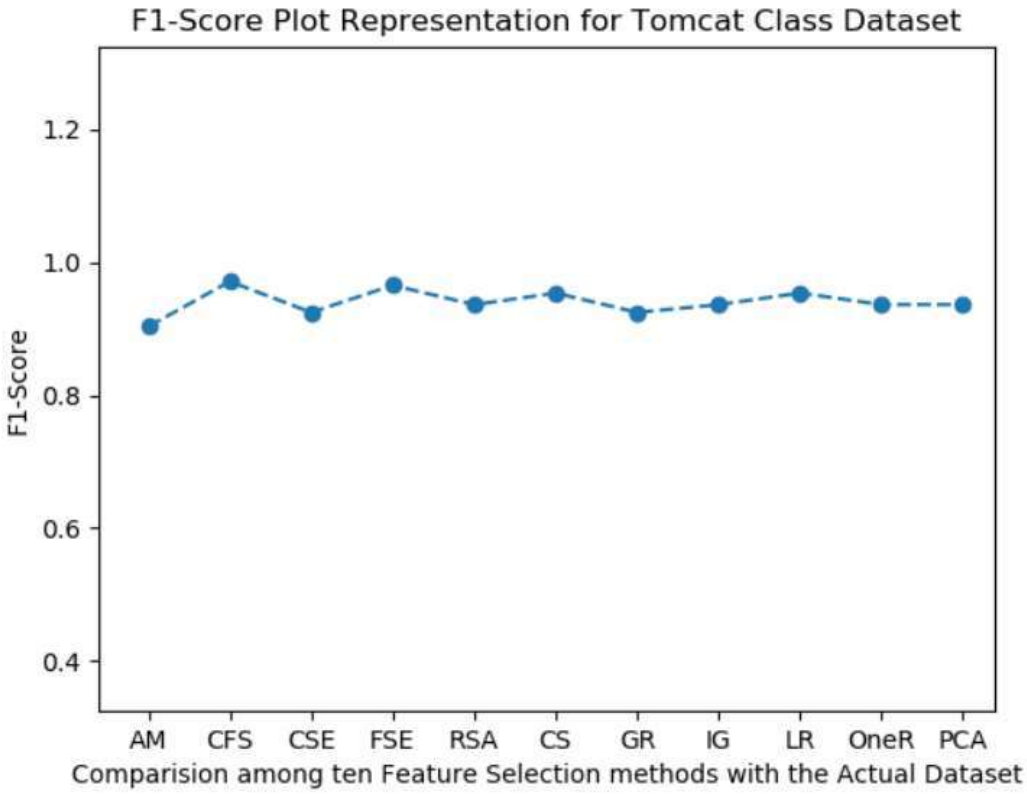

Fig 7.4: Line representation of F1-Scores for Tomcat-Class dataset

F1-Scores of all comparisons:

Dataset Name	F1-Scores across all kernels
Tomcat_AM	0.903
Tomcat_CS	0.953
Tomcat_GR	0.924
Tomcat_IG	0.936
Tomcat_LR	0.953
Tomcat_OneR	0.936
Tomcat_PCA	0.936
Tomcat_CFS	0.97
Tomcat_CSE	0.924
Tomcat_FSE	0.965
Tomcat_RSA	0.936

For the comparison of F1-Scores of all the methods including the actual dataset across all the kernels, the maximum f1-score in the Tomcat dataset is recorded in the case of CFS Subset Evaluation(0.97) while the minimum f1-score is recorded in the case of its actual dataset(0.903).

Comparisons of faulty classes across all the three kernels for Ant-Class dataset

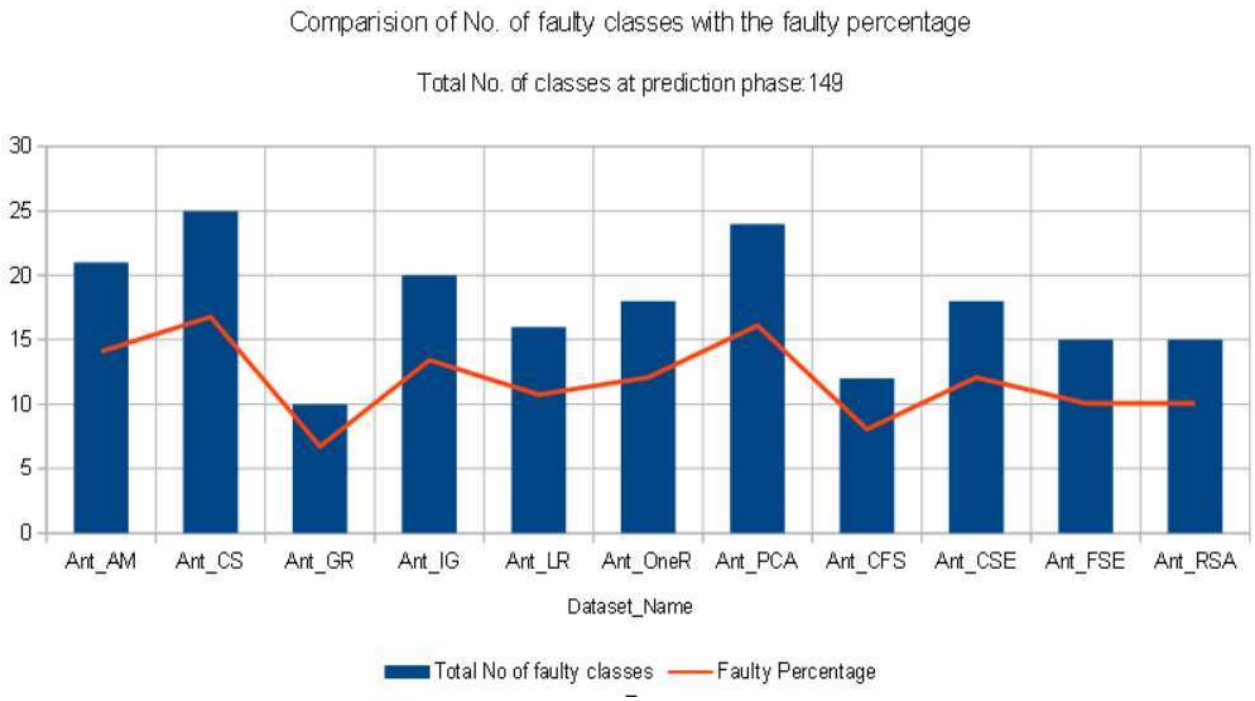

Fig 7.5: Bar-Line representation of faulty classes and faulty percentage for Ant-Class dataset

Number of faulty classes and faulty percentage of all comparisons:

Dataset Name	Total No of Classes in prediction phase	Total No of faulty classes	Faulty Percentage
Ant_AM	149	21	14.094
Ant_CS	149	25	16.779
Ant_GR	149	10	6.711
Ant_IG	149	20	13.423
Ant_LR	149	16	10.738
Ant_OneR	149	18	12.081
Ant_PCA	149	24	16.107
Ant_CFS	149	12	8.054
Ant_CSE	149	18	12.081
Ant_FSE	149	15	10.067
Ant_RSA	149	15	10.067

For the comparison of faulty percentages of all the methods including the actual dataset across all the kernels, the maximum faulty percentage in the Ant dataset is recorded in the case of Chi Square Evaluation(16.779%) while the minimum faulty percentage is recorded in the case of Gain Ratio Evaluation(6.711%).

Comparisons of faulty classes across all the three kernels for Tomcat-Class dataset

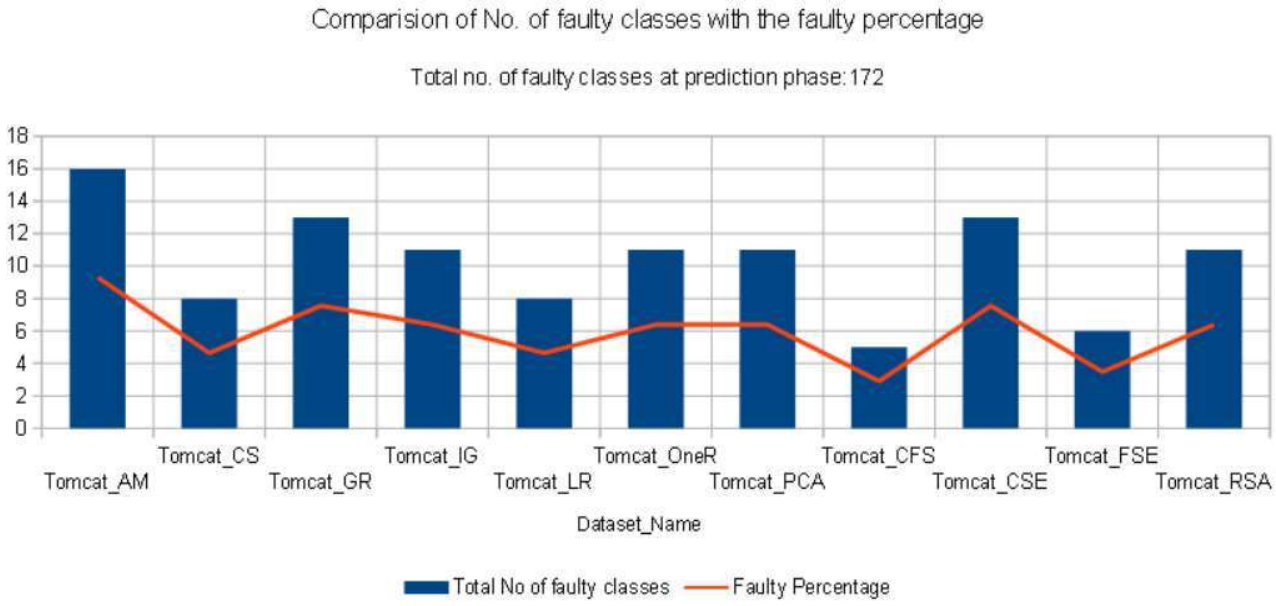

Fig 7.6: Bar-Line representation of faulty classes and faulty percentage for Tomcat-Class dataset

Number of faulty classes and faulty percentage of all comparisons:

Dataset Name	Total No of Classes in prediction phase	Total No of faulty classes	Faulty Percentage
Tomcat_AM	172	16	9.302
Tomcat_CS	172	8	4.651
Tomcat_GR	172	13	7.558
Tomcat_IG	172	11	6.395
Tomcat_LR	172	8	4.651
Tomcat_OneR	172	11	6.395
Tomcat_PCA	172	11	6.395
Tomcat_CFS	172	5	2.907
Tomcat_CSE	172	13	7.558
Tomcat_FSE	172	6	3.488

For the comparison of faulty percentages of all the methods including the actual dataset across all the kernels, the maximum faulty percentage in the Ant dataset is recorded in the case of its actual dataset(9.302%) while the minimum faulty percentage is recorded in the case of CFS Subset Evaluation(2.907%).

Screenshots

Documented results

We have considered all metrics of Tomcat-Class dataset to demonstrate the working result.

Least Square Support Vector Machine (LSSVM) for Software Defect Forecasting

```
C:\Users\Lenovo\AppData\Local\Programs\Python\Python38\python.exe C:/Users/Lenovo
/IdeaProjects/Projectify/datasetsEval_tomcat.py

##################################################

        All Metrics Evaluation

##################################################

Dataset:      Features.shape:    No. of classes:
Tomcat         (858, 20)              6

##################################################

Tomcat

Linear Kernel
---------- EVALUATION OF MODEL ----------

K-Fold(10 splits) Cross Validation Score: 94.35016835016835
Evaluation-Accuracy Score=  91.6058394160584
Evaluation-F1 Score(micro)= 0.9452554744525548
Confusion_Matrix =
 [[502   0   0   0   0]
 [ 30   0   0   0   0]
 [ 12   0   0   0   0]
 [  1   0   0   0   0]
 [  3   0   0   0   0]]
Classification =
              precision    recall  f1-score   support

           0       0.92      1.00      0.96       502
           1       0.00      0.00      0.00        30
           2       0.00      0.00      0.00        12
           3       0.00      0.00      0.00         1
           4       0.00      0.00      0.00         3

    accuracy                           0.92       548
   macro avg       0.18      0.20      0.19       548
weighted avg       0.84      0.92      0.88       548

---------- VALIDATION OF MODEL ----------

K-Fold(10 splits) Cross Validation Score: 90.54945054945055
Validation-Accuracy Score=  92.02898550724638
Validation-F1 Score(micro)= 0.9347826086956522
Confusion_Matrix =
 [[127   0   0   0]
 [  9   0   0   0]
 [  1   0   0   0]
 [  1   0   0   0]]
```

Least Square Support Vector Machine (LSSVM) for Software Defect Forecasting

```
Classification =
              precision    recall  f1-score   support

           0       0.92      1.00      0.96       127
           1       0.00      0.00      0.00         9
           2       0.00      0.00      0.00         1
           4       0.00      0.00      0.00         1

    accuracy                           0.92       138
   macro avg       0.23      0.25      0.24       138
weighted avg       0.85      0.92      0.88       138

---------- PREDICTION OF MODEL ----------

Predicted Labels:
 [0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 4 1 0 0 0 0 0 4]
Actual Labels:
 [0 0 1 0 0 1 0 0 6 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0
 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 1 2 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1]

K-Fold(10 splits) Cross Validation Score: 87.77777777777777
Prediction-Accuracy Score=  88.37209302325581
Prediction-F1 Score(micro)= 0.9069767441860465
Confusion_Matrix =
 [[152   0   0   0   0]
 [ 16   0   0   0   0]
 [  2   0   0   0   0]
 [  1   0   0   0   0]
 [  1   0   0   0   0]]
Classification =
              precision    recall  f1-score   support

           0       0.88      1.00      0.94       152
           1       0.00      0.00      0.00        16
           2       0.00      0.00      0.00         2
           4       0.00      0.00      0.00         1
           6       0.00      0.00      0.00         1

    accuracy                           0.88       172
   macro avg       0.18      0.20      0.19       172
weighted avg       0.78      0.88      0.83       172
```

---------- COST PREDICTION FOR THE MODEL ----------
(Considered coefficients for Minimum removal costs and Maximum fault identification efficiencies):

For Linear Kernel

Estimation Cost = 49.843199999999996
No of Faulty Classes: 16
Total No of Classes: 172
Percentage of Faulty Classes: 9.30232558139535
Total Cost= 24.921599999999998
Normalized Estimation Cost= 2.0
Requires performing of testing

For Ant-Class dataset:

---------- CONCLUSION OF MODEL ----------

Fault Prediction Useful:
['Ant_Consistency_Polynomial', 'Ant_OneR_RBF']

Fault Prediction requires performing of testing:
['Ant_Linear', 'Ant_Polynomial', 'Ant_RBF', 'Ant_CFS_Linear', 'Ant_CFS_Polynomial', '
Ant_CFS_RBF', 'Ant_Consistency_Linear', 'Ant_Consistency_RBF', 'Ant_Filtered_Linear
', 'Ant_Filtered_Polynomial', 'Ant_Filtered_RBF', 'Ant_RSA_Linear', '
Ant_RSA_Polynomial', 'Ant_RSA_RBF', 'Ant_ChiSq_Linear', 'Ant_ChiSq_Polynomial', '
Ant_ChiSq_RBF', 'Ant_GainR_Linear', 'Ant_GainR_Polynomial', 'Ant_GainR_RBF', '
Ant_InfoGain_Linear', 'Ant_InfoGain_Polynomial', 'Ant_InfoGain_RBF', '
Ant_LogReg_Linear', 'Ant_LogReg_Polynomial', 'Ant_LogReg_RBF', 'Ant_OneR_Linear
', 'Ant_OneR_Polynomial', 'Ant_PCA_Linear', 'Ant_PCA_Polynomial', 'Ant_PCA_RBF']

Process finished with exit code 0

For Tomcat-Class dataset:

---------- CONCLUSION OF MODEL ----------

Fault Prediction Useful:
['Tomcat_Polynomial']

Fault Prediction requires performing of testing:
['Tomcat_Linear', 'Tomcat_RBF', 'Tomcat_CFS_Linear', 'Tomcat_CFS_Polynomial', '
Tomcat_CFS_RBF', 'Tomcat_Consistency_Linear', 'Tomcat_Consistency_Polynomial', '
Tomcat_Consistency_RBF', 'Tomcat_Linear', 'Tomcat_Polynomial', 'Tomcat_RBF', '
Tomcat_RSA_Linear', 'Tomcat_RSA_Polynomial', 'Tomcat_RSA_RBF', '
Tomcat_ChiSq_Linear', 'Tomcat_ChiSq_Polynomial', 'Tomcat_ChiSq_RBF', '
Tomcat_GainR_Linear', 'Tomcat_GainR_Polynomial', 'Tomcat_GainR_RBF', '
Tomcat_InfoGain_Linear', 'Tomcat_InfoGain_Polynomial', 'Tomcat_InfoGain_RBF', '
Tomcat_LogReg_Linear', 'Tomcat_LogReg_Polynomial', 'Tomcat_LogReg_RBF', '
Tomcat_OneR_Linear', 'Tomcat_OneR_Polynomial', 'Tomcat_OneR_RBF', '
Tomcat_PCA_Linear', 'Tomcat_PCA_Polynomial', 'Tomcat_PCA_RBF']

Process finished with exit code 0

CHAPTER 8
CONCLUSION AND FUTURE WORK

CONCLUSION AND FUTURE WORK

In this study, we have conducted a comprehensive experimental evaluation of Least Squares Support Vector Machine (LS-SVM) for classification. This study also emphasized on designing a cost-based evaluation framework for determining the efficiency of the developed fault prediction model which is created using object-oriented source code metrics. Twenty object-oriented source code metrics are used to develop a model using Least Square Support Vector Machines (LSSVM) with three different kernel methods i.e., linear kernel, polynomial kernel, and RBF kernel. We have implemented these three kernel methods across four feature subset selection methods, six feature ranking methods and with the original datasets; therefore, 11 LS-SVM functions have been tested using two different open java projects following object-oriented paradigm represented as datasets (i.e); Ant-Class dataset and Tomcat-Class dataset. The experiments conducted and results generated are performed in R, Weka and Python environment.

8.1 Conclusions:

The conclusions for the above work are as follows:

1. Our experiment results suggest that it is possible to identify a small subset of OO source code metrics. The fault prediction model developed using this identified set of OO source code metrics is able to predict faulty and non-faulty classes with better accuracy.

2. We found that the selection of the classification method to develop a fault prediction model is affected by the feature selection methods.

3. Based on cost analysis framework, it is observed that as the percentage value of faulty classes increases, the fault-prediction technique tends to have a higher value of NE_{cost} .

8.2 Future Work:

In this study, developed models for fault prediction only predict whether a class is faulty or not. Future work can be extended to indicate the possible number of bugs in the class. Further, this work can be replicated over other open-source projects which use soft computing models to achieve higher accuracy of fault prediction.

8.3 Threat to validity:

For the sake of completeness, some of the existing threats to the validity of the proposed work have been considered. The proposed work may suffer from the following threats:

8.3.1 Internal Validity:

We identify two possible threats to the internal validity of our study.

i. The first threat is the consistency of project fault data. We collect fault data of project from promise repository 5. Any fault information not mentioned in the sources were not considered in the study. We do not make claims about the accuracy of data, but we believe that the data are collected consistently.

ii. The second threat is that the cost parameter may be incorrectly collected. We take cost parameter (C_u , C_i , C_s , C_f) from Wagner (2006) and fault identification efficiencies(δ_u , δ_i , δ_s) from Jones (2010), and M_p from Huitt and Wilde (1992). However, all these values may vary with company benchmarks.

8.3.2 Construct Validity:

In this study, developed models for fault prediction only predict whether a class is faulty or not, but does not indicate a possible number of bugs in the class. Nonetheless, we did realize that this is a potential threat to the construct validity of the dependent variable (fault proneness) and needs to be eliminated in future work.

8.3.3 External Validity:

In this work, all case studies are designed in Python and R language. However, the models designed in this study are likely to be valid for other object-oriented programming languages. Further research can be extended to design a model for other programming paradigms too. A number of psychological factors also affect the reliability of the software. But in this study, factors such as different level of expertise for developers, standards in which software is developed, types of developers involved, history of the development of the system and other stockholders of the system are not considered.

CHAPTER 9
APPENDIX

9. APPENDIX

Previous studies consider a small number of software projects to investigate the relationships between fault proneness and Object-Oriented metrics. Hence it is not possible to draw generic conclusions applicable to all software projects and systems. Datasets used in this study are accumulated from the PROMISE data repository 5. The PROMISE data repository is an open-source repository, which is publicly available in http://openscience.us/repo/defect/. and https://github.com/lov505/JSS-Dataset/. . It contains fault-datasets of various open-source software projects such as Apache projects, Mozilla Firefox, and others. Currently, it contains 48 software fault datasets corresponding to three types of software metrics: CK Object-Oriented metrics, Halstead/McCabe procedural code metrics, and some other static code metrics. As, we partitioned fault dataset into a training dataset, validation dataset, and testing dataset, therefore, datasets with more than 300 software modules have been considered in the study.

In this study, two open-source software projects are considered to strengthen and generalize our conclusions. In these datasets, object-oriented (OO) metrics related to coupling (CBO, CBM, CA, CE, and IC), cohesion (LCOM, LCOM3, and CAM), inheritance (DIT, NOC, and MFA), encapsulation (DAM and MOA), complexity (WMC, Avg_CC, Max_CC, and AMC), size metrics (LOC and NPM), and RFC measures have been reported, which we have used for performing the experimentations. We fetch the experimental dataset for our work from promise repository 5. The detailed definition and explanation of used metrics can be found in Jureczko (2011). Several studies have investigated the capabilities of OO metrics for software fault prediction previously and found that OO metrics performed significantly better in predicting software faults compared to static code metric (Briand &Wüst, 2002a; Shatnawi& Li, 2008). Thus, these metrics can be used for building fault prediction models.

Here the datasets used in the project are shown below. Ant class dataset consists of 745 instances with 20 attributes along with a class label. Tomcat class dataset consists of 858 instances with 20 attributes along with a class label. The below representation of datasets is shown for the first 50 instances across the 20 attributes and the remaining instances are followed in the same way. Table 9.1 represents Ant class dataset used and Table 9.2 indicates Tomcat class dataset used for the project.

Table 9.1: Ant class dataset used for the LS-SVM model

clsname	version	name
ant	1.7	org.apache.tools.ant.taskdefs.rmic.RmicAdapterFactory
ant	1.7	org.apache.tools.ant.taskdefs.optional.perforce.FStatP4OutputHandler
ant	1.7	org.apache.tools.ant.taskdefs.optional.junit.OutErrSummaryJUnitResultFormatter
ant	1.7	org.apache.tools.ant.taskdefs.optional.perforce.P4HandlerAdapter
ant	1.7	org.apache.tools.ant.taskdefs.WaitFor
ant	1.7	org.apache.tools.ant.taskdefs.optional.depend.constantpool.ConstantCPInfo
ant	1.7	org.apache.tools.zip.AsiExtraField
ant	1.7	org.apache.tools.ant.types.resources.selectors.Date
ant	1.7	org.apache.tools.ant.util.facade.FacadeTaskHelper
ant	1.7	org.apache.tools.ant.types.selectors.DepthSelector
ant	1.7	org.apache.tools.ant.taskdefs.optional.dotnet.JSharp
ant	1.7	org.apache.tools.ant.taskdefs.LogStreamHandler
ant	1.7	org.apache.tools.ant.input.InputHandler
ant	1.7	org.apache.tools.zip.JarMarker
ant	1.7	org.apache.tools.ant.taskdefs.optional.clearcase.CCMkbl
ant	1.7	org.apache.tools.ant.types.resources.Restrict
ant	1.7	org.apache.tools.ant.DemuxInputStream
ant	1.7	org.apache.tools.ant.taskdefs.optional.ejb.IPlanetEjbc
ant	1.7	org.apache.tools.zip.ZipExtraField
ant	1.7	org.apache.tools.ant.types.resources.comparators.FileSystem
ant	1.7	org.apache.tools.ant.taskdefs.condition.TypeFound
ant	1.7	org.apache.tools.ant.taskdefs.optional.perforce.P4OutputHandler
ant	1.7	org.apache.tools.ant.taskdefs.email.Message
ant	1.7	org.apache.tools.ant.taskdefs.optional.vss.MSVSSCP
ant	1.7	org.apache.tools.ant.taskdefs.condition.IsSet
ant	1.7	org.apache.tools.ant.taskdefs.optional.Rpm
ant	1.7	org.apache.tools.ant.types.selectors.BaseExtendSelector
ant	1.7	org.apache.tools.ant.taskdefs.optional.junit.XMLJUnitResultFormatter
ant	1.7	org.apache.tools.ant.taskdefs.optional.net.SetProxy
ant	1.7	org.apache.tools.ant.util.KeepAliveInputStream
ant	1.7	org.apache.tools.ant.taskdefs.CallTarget
ant	1.7	org.apache.tools.ant.taskdefs.DefBase
ant	1.7	org.apache.tools.zip.ZipOutputStream
ant	1.7	org.apache.tools.ant.taskdefs.optional.perforce.P4Label
ant	1.7	org.apache.tools.ant.taskdefs.XSLTLiaison
ant	1.7	org.apache.tools.ant.types.resources.BCFileSet
ant	1.7	org.apache.tools.ant.taskdefs.optional.extension.ExtraAttribute
ant	1.7	org.apache.tools.ant.taskdefs.optional.vss.MSVSS
ant	1.7	org.apache.tools.ant.util.regexp.Jdk14RegexpRegexp
ant	1.7	org.apache.tools.ant.taskdefs.condition.AntVersion
ant	1.7	org.apache.tools.ant.util.LazyFileOutputStream
ant	1.7	org.apache.tools.ant.util.LoaderUtils
ant	1.7	org.apache.tools.ant.loader.AntClassLoader2
ant	1.7	org.apache.tools.ant.taskdefs.rmic.ForkingSunRmic
ant	1.7	org.apache.tools.ant.util.StringUtils
ant	1.7	org.apache.tools.ant.types.Parameter
ant	1.7	org.apache.tools.ant.util.regexp.Jdk14RegexpMatcher
ant	1.7	org.apache.tools.ant.DirectoryScanner
ant	1.7	org.apache.tools.ant.taskdefs.LogOutputStream
ant	1.7	org.apache.tools.ant.taskdefs.optional.jsp.JspMangler

wmc	dit	noc	cbo	rfc	lcom	ca	ce	npm	lcom3
3	1	0	10	18	3	1	9	1	1.1
5	2	0	4	13	0	1	4	4	0.625
1	2	0	1	3	0	0	1	1	2
8	1	9	13	20	12	9	4	8	0.8
9	3	0	5	26	16	0	5	7	0.75
3	2	5	7	4	1	6	1	2	0.5
20	1	0	4	40	130	0	4	18	0.737
13	1	0	7	28	20	2	5	12	0.810
9	1	0	5	19	8	4	1	9	0.531
7	5	0	9	25	0	6	3	7	0.667
9	6	0	5	17	26	0	5	6	0.833
3	2	0	19	10	3	14	5	3	2
1	1	0	10	1	0	8	2	1	2
9	1	0	3	12	20	1	2	8	0.813
19	4	0	7	40	129	0	7	16	0.937
10	4	2	13	26	0	7	7	9	0.111
3	2	0	2	5	0	1	1	3	0
20	1	0	7	79	136	5	5	10	0.898
6	1	0	8	6	15	7	1	6	2
3	4	0	5	11	1	1	4	1	0.5
5	2	0	7	14	6	0	7	4	0.625
1	1	0	1	1	0	0	1	1	2
11	2	0	7	26	0	5	2	11	0.58
3	4	0	5	15	3	0	5	2	2
3	2	0	5	7	1	1	4	3	0.5
16	3	0	15	65	76	0	15	13	0.92
4	4	8	12	5	0	8	4	3	0.333
15	1	0	11	65	75	0	11	11	0.694
11	3	0	3	24	17	0	3	10	0.671
2	3	0	1	3	1	1	0	2	2
15	3	0	9	41	0	0	9	15	0.679
14	4	3	11	29	85	3	8	11	0.577
27	3	0	8	101	157	5	3	15	0.825
5	4	0	8	23	4	4	8	5	0.75
3	1	0	2	3	3	2	0	3	1.5
4	5	0	5	19	6	0	5	4	2
6	1	0	2	11	3	1	1	4	0.6
55	3	8	20	106	1313	8	12	9	0.957
3	2	0	4	19	3	0	4	2	2
8	1	0	4	22	0	0	4	6	0.429
11	2	0	1	23	39	1	0	10	0.483
10	1	0	10	29	43	7	3	8	0.833
1	3	0	1	2	0	0	1	1	2
3	2	1	12	22	3	2	10	2	1.5
9	1	0	32	36	36	32	0	7	1.104
7	1	0	24	8	3	24	0	7	0.667
9	1	1	4	19	30	1	3	7	0.5
63	1	2	61	144	1603	51	10	31	0.874
6	3	2	20	13	7	17	3	3	0.4
2	1	0	6	2	1	6	0	2	2

loc	dam	moa	mfa	cam	ic	cbm	amc	max_cc	avg_cc	bug
106	0	0	0	0.444	0	0	32.6667	1	0.667	0
76	1	1	0.7	0.5	0	0	13.4	1	0.6	0
7	0	0	1	1	0	0	6	0	0	0
101	0.2	1	0	0.406	0	0	11	1	0.875	0
185	1	0	0.8	0.389	0	0	19	2	1	1
16	1	0	0.75	0.556	0	0	4	1	0.667	0
345	1	1	0	0.284	0	0	15.9	3	1.15	1
183	1	2	0	0.3	0	0	12.5385	7	1.385	0
119	1	0	0	0.519	0	0	11.7778	3	1.444	0
255	0	0	0.864	0.343	2	5	34.8571	9	3.143	0
71	0	0	0.949	0.333	1	4	6.556	3	1.111	0
38	0	0	0.923	0.583	0	0	11.6667	1	0.333	0
1	0	0	0	1	0	0	0	1	1	0
54	1	3	0	0.375	0	0	4.556	1	0.778	0
252	0.5	0	0.727	0.382	2	3	11.5263	6	1.316	0
110	1	1	0.795	0.26	1	1	9.9	3	1.6	0
34	1	1	0.818	0.5	1	2	10	1	0.667	0
835	1	1	0	0.194	0	0	39.8	10	1.95	0
6	0	0	0	0.444	0	0	0	1	1	0
41	1	1	0.971	0.75	1	2	12.3333	3	1	1
70	1	0	0.692	0.8	0	0	12.6	4	1.4	0
1	0	0	0	1	0	0	0	1	1	0
181	1	0	0.529	0.409	0	0	15	1	0.727	0
51	0	0	0.978	0.667	1	1	16	2	1	0
29	1	0	0.818	0.667	0	0	8.333	1	0.667	0
483	1	0	0.725	0.311	0	0	28.25	9	1.563	1
18	1	1	0.921	0.438	0	0	3.25	1	0.75	0
443	1	0	0	0.295	0	0	28.0667	5	1.6	0
309	1	0	0.787	0.545	0	0	26.4545	11	1.909	0
7	0	0	0.947	0.75	1	1	2.5	1	0.5	0
204	1	1	0.725	0.225	1	2	12.3333	2	1.533	0
108	1	1	0.759	0.321	0	0	6.571	4	1.286	0
1286	0.889	1	0.294	0.184	1	7	45.6296	4	1.111	1
249	1	0	0.927	0.8	2	2	48.2	1	0.8	0
4	0	0	0	0.667	0	0	0	1	1	0
57	0	0	0.978	0.625	2	4	13.25	2	1	0
53	1	0	0	0.667	0	0	7.5	1	0.833	0
1133	1	2	0.407	0.195	0	0	19.1273	10	1.855	0
136	0	0	0.8	0.667	0	0	44.3333	2	1	0
169	1	0	0	0.625	0	0	19.875	7	1.625	0
136	1	0	0.455	0.348	1	1	10.8182	1	0.546	0
91	0.5	1	0	0.2	0	0	7.9	3	1.2	0
4	0	0	1	1	0	0	3	0	0	0
72	0	0	0.867	1	0	0	22.6667	1	0.667	0
281	0.833	0	0	0.25	0	0	29.5556	11	3	0
43	1	0	0	0.714	0	0	4.714	1	0.857	0
130	1	0	0	0.667	0	0	13.3333	4	1.222	0
2303	1	2	0	0.214	0	0	35.0159	35	3.238	3
56	1	1	0.733	0.433	1	4	8	1	0.667	0
2	0	0	0	0.667	0	0	0	1	1	0

Table 9.2: Tomcat class dataset used for the LS-SVM model

clsname	version	name
Tomcat	6.0.389418	org.apache.coyote.http11.filters.VoidOutputFilter
Tomcat	6.0.389418	org.apache.el.parser.AstGreaterThan
Tomcat	6.0.389418	org.apache.coyote.Request
Tomcat	6.0.389418	javax.el.MethodNotFoundException
Tomcat	6.0.389418	org.apache.naming.EjbRef
Tomcat	6.0.389418	org.apache.jasper.compiler.Compiler
Tomcat	6.0.389418	javax.servlet.ServletContextAttributeListener
Tomcat	6.0.389418	org.apache.el.MethodExpressionLiteral
Tomcat	6.0.389418	org.apache.naming.ResourceEnvRef
Tomcat	6.0.389418	org.apache.catalina.mbeans.StandardServerMBean
Tomcat	6.0.389418	org.apache.naming.NamingService
Tomcat	6.0.389418	org.apache.catalina.startup.WebRuleSet
Tomcat	6.0.389418	org.apache.naming.resources.ImmutableNameNotFoundException
Tomcat	6.0.389418	javax.servlet.jsp.tagext.JspTag
Tomcat	6.0.389418	org.apache.tomcat.util.net.jsse.JSSE13SocketFactory
Tomcat	6.0.389418	org.apache.catalina.realm.MemoryRuleSet
Tomcat	6.0.389418	org.apache.tomcat.jni.Pool
Tomcat	6.0.389418	javax.servlet.jsp.JspEngineInfo
Tomcat	6.0.389418	org.apache.jk.common.HandlerRequest
Tomcat	6.0.389418	org.apache.coyote.http11.Http11Processor
Tomcat	6.0.389418	org.apache.catalina.realm.Constants
Tomcat	6.0.389418	org.apache.el.parser.AstAnd
Tomcat	6.0.389418	org.apache.catalina.valves.FastCommonAccessLogValve
Tomcat	6.0.389418	org.apache.naming.NamingContextEnumeration
Tomcat	6.0.389418	org.apache.jasper.compiler.SmapGenerator
Tomcat	6.0.389418	org.apache.catalina.core.ApplicationHttpRequest
Tomcat	6.0.389418	org.apache.tomcat.util.buf.B2CConverter
Tomcat	6.0.389418	javax.servlet.jsp.tagext.TagFileInfo
Tomcat	6.0.389418	org.apache.catalina.startup.SetParentClassLoaderRule
Tomcat	6.0.389418	javax.servlet.jsp.tagext.PageData
Tomcat	6.0.389418	org.apache.catalina.Authenticator
Tomcat	6.0.389418	org.apache.el.lang.ExpressionBuilder
Tomcat	6.0.389418	org.apache.catalina.ant.ListTask
Tomcat	6.0.389418	org.apache.catalina.core.Constants
Tomcat	6.0.389418	org.apache.tomcat.util.net.puretls.PureTLSSupport
Tomcat	6.0.389418	org.apache.catalina.ant.jmx.Arg
Tomcat	6.0.389418	org.apache.jk.config.GeneratorJk1
Tomcat	6.0.389418	org.apache.tomcat.jni.Poll
Tomcat	6.0.389418	org.apache.catalina.core.StandardHost
Tomcat	6.0.389418	org.apache.jasper.compiler.JasperTagInfo
Tomcat	6.0.389418	org.apache.catalina.util.StringManager
Tomcat	6.0.389418	org.apache.catalina.LifecycleException
Tomcat	6.0.389418	org.apache.tomcat.util.http.NamesEnumerator
Tomcat	6.0.389418	org.apache.tomcat.util.buf.ReadConvertor
Tomcat	6.0.389418	javax.el.ExpressionFactory
Tomcat	6.0.389418	org.apache.coyote.http11.InternalAprOutputBuffer
Tomcat	6.0.389418	org.apache.catalina.users.MemoryRole
Tomcat	6.0.389418	org.apache.catalina.Role
Tomcat	6.0.389418	org.apache.tomcat.util.digester.SetPropertiesRule
Tomcat	6.0.389418	org.apache.catalina.ant.JMXQueryTask

wmc	dit	noc	cbo	rfc	lcom	ca	ce	npm	lcom3
8	1	0	6	14	26	2	0	7	1
2	4	0	4	5	1	1	0	2	2
56	1	0	49	89	1310	40	0	56	0.956
4	5	1	0	8	6	0	0	4	2
3	2	0	3	8	3	3	0	3	1.5
16	1	2	38	91	58	23	0	13	0.778
3	1	0	0	3	3	0	0	3	2
10	3	0	3	24	3	1	0	10	0.481
3	2	0	2	7	3	2	0	3	1.5
3	2	0	5	8	3	0	0	2	1
11	2	0	1	25	35	0	0	11	0.65
4	2	0	12	22	2	1	0	4	0.583
6	5	0	1	7	15	1	0	6	2
0	1	0	0	0	0	0	0	0	2
6	3	0	8	33	3	1	0	1	0.4
3	2	0	6	9	1	2	0	3	0
14	1	0	3	15	91	3	0	14	2
2	1	1	0	3	1	0	0	2	2
22	2	0	20	145	127	0	0	15	0.867
51	1	0	40	207	999	4	0	38	0.907
1	1	0	0	2	0	0	0	1	2
2	4	0	4	6	1	1	0	2	2
33	2	0	9	93	378	0	0	26	0.901
6	1	0	4	12	3	3	0	6	0.2
8	1	0	2	31	12	1	0	8	0.457
37	3	0	8	97	508	2	0	21	0.895
8	1	1	11	21	10	5	0	4	0.735
4	1	0	0	5	0	0	0	4	0.667
2	2	0	5	8	0	1	0	2	0
2	1	1	0	3	1	0	0	2	2
0	1	0	3	0	0	3	0	0	2
9	1	0	19	46	8	3	0	5	0.438
2	5	0	2	5	1	0	0	2	2
1	1	0	0	2	0	0	0	1	2
7	1	0	9	36	0	1	0	5	0.417
6	1	0	2	7	7	2	0	6	0.7
11	1	0	2	23	15	1	0	11	0.7
10	1	0	4	11	45	3	0	10	1.111
42	2	0	26	98	683	10	0	41	0.948
3	2	0	2	4	0	2	0	3	0
10	1	0	79	37	39	76	0	7	0.593
7	3	0	35	12	11	35	0	7	0.167
4	1	0	2	9	0	1	0	2	0.25
4	3	0	2	6	6	1	0	4	1
5	1	1	0	6	10	0	0	5	2
30	1	0	14	71	135	2	0	22	0.781
3	2	0	3	9	1	1	0	2	0.5
5	1	0	18	5	10	18	0	5	2
7	2	0	4	25	9	1	0	7	0.5
4	5	0	2	11	0	0	0	4	0

loc	dam	moa	mfa	cam	ic	cbm	amc	max_cc	avg_cc	bug
39	1	2	0	0.393	0	0	3.5	1	0.75	0
36	0	0	0.976	0.667	1	1	17	1	0.5	0
583	1	24	0	0.138	0	0	8.821	5	1.196	0
20	0	0	1	0.667	0	0	4	0	0	0
73	0	0	0.938	0.833	0	0	22	3	1	0
717	0.889	7	0	0.28	0	0	43.25	15	1.938	1
3	0	0	0	1	0	0	0	1	1	0
114	1	0	0.429	0.211	1	1	10.1	3	1	0
30	0	0	0.938	0.833	0	0	8.667	3	1	0
27	1	0	0.976	1	0	0	7.667	1	0.333	0
283	1	0	0.412	0.318	0	0	24.3636	2	1.182	0
1700	1	3	0.5	0.5	1	1	423	1	0.5	2
14	0	0	0.865	0.458	1	1	1.333	1	0.833	0
0	0	0	0	0	0	0	0	0	0	0
177	1	0	0.828	0.4	1	6	28.3333	1	0.833	0
35	1	0	0.667	0.556	1	1	10.3333	1	0.333	0
17	0	0	0	0.286	0	0	0.214	1	0.929	0
5	0	0	0	1	0	0	1.5	1	0.5	0
1226	0.857	1	0.545	0.306	2	2	54.0909	10	2.318	0
2594	1	9	0	0.092	0	0	49.1569	41	4.137	4
10	0	0	0	1	0	0	3	0	0	0
33	0	0	0.976	0.667	1	1	15.5	1	0.5	0
995	1	2	0.392	0.188	0	0	28.2727	6	1.485	0
40	1	0	0	0.583	0	0	5.5	1	0.833	0
259	1	0	0	0.325	0	0	30.75	5	1.5	0
981	1	3	0.620	0.242	2	3	25.0541	11	2.757	0
108	0.714	2	0	0.429	0	0	11.625	2	0.75	1
28	1	1	0	0.5	0	0	5.25	1	0.75	0
32	0	0	0.917	0.625	0	0	14.5	1	0.5	0
5	0	0	0	1	0	0	1.5	1	0.5	0
0	0	0	0	0	0	0	0	0	0	0
350	1	3	0	0.313	0	0	37.4444	1	0.778	0
11	0	0	0.986	1	1	1	4.5	1	0.5	0
9	0	0	0	1	0	0	3	0	0	1
181	0.5	0	0	0.444	0	0	24.5714	1	0.714	0
29	0	0	0	0.75	0	0	3.5	1	0.833	0
167	0	1	0	0.318	0	0	13.7273	2	1	0
24	0	0	0	0.34	0	0	0.3	1	0.9	0
962	1	0	0.633	0.25203	2	7	21.5	10	1.548	0
31	1	0	0.895	0.444	0	0	9	2	1	0
296	1	0	0	0.611	0	0	28.3	5	1.5	0
77	1	0	0.85	0.524	1	2	9.714	4	0.857	0
91	0	1	0	0.625	0	0	20.75	5	2	0
21	1	1	0.833	0.4	1	2	4	1	0.75	0
8	0	0	0	0.533	0	0	0.6	1	0.8	0
836	1	6	0	0.162	0	0	26.4	6	1.633	0
59	1	1	0.75	0.556	0	0	18.3333	2	1	0
5	0	0	0	0.7	0	0	0	1	1	0
331	1	0	0.786	0.343	2	2	46	5	1	0
56	1	0	0.958	0.625	2	2	12.75	1	0.75	0

CHAPTER 10
BIBILOGRAPHY

10. BIBILOGRAPHY

Kumar, L. , Krishna, A. , Rath, S.K. , 2016b. The impact of feature selection on maintainability prediction of service-oriented applications. Serv. Oriented Computer. Appl. .

Suykens, J.A. , De Brabanter, J. , Lukas, L. , Vandewalle, J. , 2002. Weighted least squares support vector machines: robustness and sparse approximation. Neurocomputing .

Chidamber, S.R. , Kemerer, C.F. , 1994. A metrics suite for object oriented design. IEEE Trans. Software Eng..

Martin, R. , 1994. Oo design quality metrics. Analy. Dependencies .

Bansiya, J. , Davis, C.G. , 2002. A hierarchical model for object-oriented design quality assessment. IEEE Trans. Software Eng. .

Henderson-Sellers, B., 1996. Software metrics.

Halstead, M.H. , 1977. Elements of Software Science, 7. Elsevier New York .

Tang, M.-H. , Kao, M.-H. , Chen, M.-H. , 1999. An empirical study on object-oriented metrics. In: Software Metrics Symposium, 1999. Proceedings. Sixth Interna- tional, IEEE .

McCabe, T.J. , 1976. A complexity measure. IEEE Trans. Software Eng..

Li, W. , Henry, S. , 1993. Maintenance metrics for the object oriented paradigm. In: Software Metrics Symposium, 1993. Proceedings., First International, IEEE .

Wagner, S. , 2006. A literature survey of the quality economics of defect-detection techniques. In: Proceedings of the ACM/IEEE International Symposium on Em- pirical Software Engineering (ISESE).

Briand, L.C. , Wüst, J. , Daly, J.W. , Porter, D.V. , 20 0 0. Exploring the relationships between design measures and software quality in object-oriented systems. J. Syst. Software.

Huitt, R. , Wilde, N. , 1992. Maintenance support for object-oriented programs. IEEE Trans. Software Eng.

Jones, C. , 2010. Software quality in 2010: a survey of the state of the art. Founder and Chief Scientist Emeritus .

Kohavi, R. , John, G.H. , 1997. Wrappers for feature subset selection. Artif. Intell. .
Kohavi, R. , et al. , 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Ijcai, Vol. 14.

Plackett, R.L. , 1983. Karl pearson and the chi-squared test. Int. Stat. Rev./Revue Internationale de Statistique.

Novakovic, J. , 2010. The Impact of Feature Selection on the Accuracy of naïve Bayes Classifier. In: 18th Telecommunications forum TELFOR, Vol. 2.

Cruz, A.E.C. , Ochimizu, K. ,2009. Towards logistic regression models for predicting fault-prone code across software projects. In: Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement. IEEE Computer Society.

Wang, D. , Romagnoli, J. , 2005. Robust multi-scale principal components analysis with applications to process monitoring. J. Process Control .

Shatnawi, R. , Li, W. , 2008. The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process. J. Syst. Software.

Pawlak, Z. , 1982. Rough sets. Int. J. Comput. Inf. Sci. .
Dash, M. , Liu, H. , 2003. Consistency-based search in feature selection. Artif. Intell.

Huang, S.-Y. , 1992. Intelligent Decision Support: Handbook of Applications and Advances of the Rough Sets Theory, 11. Springer Science & Business Media .

Cartwright, M. , Shepperd, M. , 20 0 0. An empirical investigation of an object-oriented software system. IEEE Trans. Software Eng. .