

**BAŞKENT UNIVERSITY
INSTITUTE OF SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER ENGINEERING
MASTER OF SCIENCE IN COMPUTER ENGINEERING**

**SOFTWARE QUALITY PREDICTION MODELS: A COMPARATIVE
INVESTIGATION BASED ON MACHINE LEARNING TECHNIQUES
FOR OBJECT-ORIENTED SYSTEMS**

BY

ÖZCAN İLHAN

MASTER OF SCIENCE THESIS

ANKARA - 2020

**BAŞKENT UNIVERSITY
INSTITUTE OF SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER ENGINEERING
MASTER OF SCIENCE IN COMPUTER ENGINEERING**

**SOFTWARE QUALITY PREDICTION MODELS: A COMPARATIVE
INVESTIGATION BASED ON MACHINE LEARNING TECHNIQUES
FOR OBJECT-ORIENTED SYSTEMS**

BY

ÖZCAN İLHAN

MASTER OF SCIENCE THESIS

ADVISOR

Asst. Prof. Dr. Tülin ERÇELEBİ AYYILDIZ

ANKARA - 2020

BAŞKENT UNIVERSITY
INSTITUTE OF SCIENCE AND ENGINEERING

This study, which was prepared by Özcan İLHAN, for the program of Master of Science in Computer Engineering, has been approved in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE in Computer Engineering Department by following committee.

Date of Thesis Defense: 27 / 08 / 2020

Thesis Title: Software Quality Prediction Models: A Comparative Investigation Based on Machine Learning Techniques for Object-Oriented Systems

Examining Committee Members

Signature

Asst. Prof. Dr. Tülin ERÇELEBİ AYYILDIZ, Başkent University

.....

Asst. Prof. Dr. Duygu DEDE ŞENER, Başkent University

.....

Asst. Prof. Dr. Damla TOPALLI, Atılım University

.....

APPROVAL

Prof. Dr. Faruk ELALDI

Director, Institute of Science and Engineering

Date: ... / ... / 2020

BAŞKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
YÜKSEK LİSANS TEZ ÇALIŞMASI ORJİNALLİK RAPORU

Tarih: 04 / 09 / 2020

Öğrencinin Adı, Soyadı : Özcan İLHAN

Öğrencinin Numarası : 21810032

Anabilim Dalı : Bilgisayar Mühendisliği

Programı : Tezli Yüksek Lisans

Danışmanın Unvanı/Adı, Soyadı : Dr. Öğr. Üyesi Tülin ERÇELEBİ AYYILDIZ

Tez Başlığı : Software Quality Prediction Models: A Comparative Investigation Based on Machine Learning Techniques for Object-Oriented Systems

Yukarıda başlığı belirtilen Yüksek Lisans tez çalışmamın; Giriş, Ana Bölümler ve Sonuç Bölümünden oluşan, toplam 81 sayfalık kısmına ilişkin, 04/09/2020 tarihinde tez danışmanım tarafından Turnitin adlı intihal tespit programından aşağıda belirtilen filtrelemeler uygulanarak alınmış olan orijinallik raporuna göre, tezimin benzerlik oranı %6'dır. Uygulanan filtrelemeler:

1. Kaynakça hariç
2. Alıntılar hariç
3. Beş (5) kelimedenden daha az örtüşme içeren metin kısımları hariç

“Başkent Üniversitesi Enstitüleri Tez Çalışması Orijinallik Raporu Alınması ve Kullanılması Usul ve Esaslarını” inceledim ve bu uygulama esaslarında belirtilen azami benzerlik oranlarına tez çalışmamın herhangi bir intihal içermediğini; aksinin tespit edileceği muhtemel durumda doğabilecek her türlü hukuki sorumluluğu kabul ettiğimi ve yukarıda vermiş olduğum bilgilerin doğru olduğunu beyan ederim.

Öğrenci İmzası:

ONAY

Tarih: 04 / 09 / 2020

Öğrenci Danışmanı Unvan, Ad, Soyad, İmza:

Dr. Öğr. Üyesi Tülin ERÇELEBİ AYYILDIZ

ACKNOWLEDGEMENTS

I would like to thank to my advisor Asst. Prof. Dr. Tülin ERÇELEBİ AYYILDIZ for her advices and guidance. Her valuable advices helped me during the whole period of my research.

I would also like to thank my family for supporting me throughout writing this thesis and my studies.

ABSTRACT

Özcan İLHAN

**SOFTWARE QUALITY PREDICTION MODELS: A COMPARATIVE
INVESTIGATION BASED ON MACHINE LEARNING TECHNIQUES FOR
OBJECT-ORIENTED SYSTEMS**

Başkent University Institute of Science

The Department of Computer Engineering

2020

The purpose of this thesis study is investigating correlation between Chidamber and Kemerer (CK) Object-Oriented (OO) software metrics and determining the accuracy rate in software bug prediction. For this reason, eleven most frequently used Machine Learning (ML) techniques and two Support Vector Machine (SVM) libraries performance was analyzed in order to find the best technique for 33 latest version of open source projects. In this thesis study, the relation between CK metrics and reliability is also determined. Each technique was evaluated using RapidMiner and WEKA tools. Dataset was validated with a 10-fold cross-validation technique. Furthermore, Bayesian belief's networks form used for determining which CK metric are primary estimators. Receiver Operating characteristic (ROC), Precision, Accuracy, Area Under the Curve (AUC), Mean Absolute Error (MAE), Root Mean Squared Error (RMSE) performance metrics used for evaluation of results. Results of this study show that Random Forrest, Bagging, AdaBoost ML techniques are the most effective for prediction models in terms of AUC values. In contrast, SVMs are the least effective models. This thesis study also revealed that Weighted Methods per class (WMC) is the most effective software metric. Then, the Number of Children (NOC), Depth of Inheritance Tree (DIT) metrics are good contributor for determining the quality of software.

KEYWORDS: Machine Learning, Object-Oriented Metrics, Software Reliability, Software Quality Metrics, Bug Prediction

Advisor: Asst. Prof. Dr. Tülin ERÇELEBİ AYYILDIZ, Başkent University, Department of Computer Engineering.

ÖZET

Özcan İLHAN

YAZILIM KALİTE TAHMİN MODELLERİ: NESNE ODAKLI SİSTEMLER İÇİN MAKİNE ÖĞRENME TEKNİKLERİNE DAYALI KARŞILAŞTIRMALI BİR ARAŞTIRMA

Başkent Üniversitesi Fen Bilimleri Enstitüsü

Bilgisayar Mühendisliği Anabilim Dalı

2020

Bu tez çalışmasının amacı, Chidamber ve Kemerer (CK) Nesne Yönelimli yazılım metrikleri arasındaki ilişkiyi araştırmak ve yazılım hata tahmininde doğruluk oranını belirlemektir. Bu nedenle, açık kaynaklı 33 projenin en son sürümleri için en iyi tekniği bulmak amacıyla en sık kullanılan 11 Makine Öğrenimi tekniği ve 2 Destek Vektör Makinesi kütüphanesinin performansı analiz edilmiştir. Bu tez çalışmasında, CK ölçütleri ile güvenilirlik arasındaki ilişki de belirlenmiştir. Her teknik RapidMiner ve WEKA araçları kullanılarak değerlendirilmiştir. Veri kümesi, 10 kat çapraz doğrulama tekniği ile doğrulanmıştır. Ayrıca, Bayesian ağları, hangi CK metriğinin en iyi tahmin edici olduğunu belirlemek için kullanılan formlardır. Sonuçların değerlendirilmesi için Alıcı İşletim Karakteristiği, Kesinlik, Doğruluk, Eğri Altında Kalan Alan, Ortalama Mutlak Hata, Ortalama Hata Kare Kökü performans ölçütleri kullanılmıştır. Bu çalışmanın sonuçları, Rastgele Orman, Torbalama, Arttırma makine öğrenmesi tekniklerinin tahmin modelleri için en etkili olduğunu göstermektedir. Buna tersine, Destek Vektör Makineleri en az etkili modellerdir. Bu tez çalışması ayrıca, Sınıfın Ağırlıklı Metot Sayısının en etkili yazılım metriği olduğunu ortaya çıkarmıştır. Daha sonra, Alt Sınıf Sayısı, Kalıtım Ağacının Derinliği ölçümleri yazılımın kalitesini belirlemede iyi bir katkı sağlar.

ANAHTAR KELİMELELER: Makine Öğrenmesi, Nesne Yönelimli Metrikler, Yazılım Güvenilirliği, Yazılım Kalite Metrikleri, Hata Tahmini

Danışman: Dr. Öğr. Üyesi Tülin ERÇELEBİ AYYILDIZ, Başkent Üniversitesi, Bilgisayar Mühendisliği Bölümü.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS.....	i
ABSTRACT.....	ii
ÖZET.....	iii
TABLE OF CONTENTS.....	iv
LIST OF TABLES.....	vii
LIST OF FIGURES.....	xi
LIST OF SYMBOLS AND ABBREVIATIONS.....	viii
1. INTRODUCTION.....	1
1.1. Problem Definition.....	1
1.2. Problem Solution.....	2
1.3. Study Motivation.....	3
1.4. Aims & Objectives.....	3
2. LITERATURE REVIEW.....	4
2.1. Previous Literature.....	4
2.2. Limitations of Previous Research.....	6
2.3. Research Questions.....	6
3. MATERIALS & METHODS.....	7
3.1. Data Collection.....	7
3.2. Software Metrics.....	9
3.3. Machine Learning Software Tools.....	11
3.3.1. WEKA machine learning software.....	11
3.3.2. RapidMiner machine learning software.....	12
4. TECHNIQUES & EXPERIMENTS.....	13
4.1. Decision Tree.....	14
4.1.1. Decision Tree analyses in WEKA.....	14
4.1.2. Decision Tree analyses in RapidMiner.....	16
4.2. Random Forest.....	18
4.2.1. Random Forest analyses in WEKA.....	19
4.2.2. Random Forest analyses in RapidMiner.....	21
4.3. Bayesian Network.....	23
4.4. Naïve Bayes.....	27
4.4.1. Naïve Bayes analyses in WEKA.....	27

4.4.1. Naïve Bayes analyses in RapidMiner.....	29
4.5. Rule based classification.....	30
4.6. Support Vector Machine.....	32
4.6.1. Sequential Minimal Optimization analyses in WEKA.....	34
4.6.2. Sequential Minimal Optimization analyses in RapidMiner.....	36
4.7. Library for Support Vector Machines.....	37
4.7.1. Library for Support Vector Machines analyses in WEKA.....	37
4.7.2. Library for Support Vector Machines analyses in RapidMiner.	39
4.8. Library for large linear classification.....	40
4.8.1. Library for large linear classification analyses in WEKA.....	41
4.8.2. Library for large linear classification analyses in RapidMiner..	44
4.9. Logistic Regression.....	44
4.9.1. Logistic Regression analyses in WEKA.....	45
4.9.2. Logistic Regression analyses in RapidMiner.....	46
4.10. Bagging.....	46
4.10.1. Bagging analyses in WEKA.....	47
4.10.2. Bagging analyses in RapidMiner.....	49
4.11. Boosting.....	51
4.11.1. Boosting analyses in WEKA.....	51
4.11.2. Boosting analyses in RapidMiner.....	53
4.12. Artificial Neural Networks.....	55
4.12.1. Artificial Neural Networks analyses in WEKA.....	57
4.12.2. Artificial Neural Networks analyses in RapidMiner.....	59
4.13. Nearest Neighbors.....	61
4.13.1. Nearest Neighbors analyses in WEKA.....	63
4.13.2. Nearest Neighbors analyses in RapidMiner.....	64
5. RESULTS.....	67
5.1. Performance Evaluation Results.....	67
5.2. Findings.....	79
6. CONCLUSION.....	81
REFERENCES.....	82

APPENDIX

APPENDIX 1: Weka Preprocess Screen

APPENDIX 2: Weka Filter Options

APPENDIX 3: Weka Knowledge Flow

APPENDIX 4: Decision Tree

APPENDIX 5: Random Forest

APPENDIX 6: Bayesian Network

APPENDIX 7: Naïve Bayes

APPENDIX 8: Rule Based Classification

APPENDIX 9: SMO Classification

APPENDIX 10: LibSVM

APPENDIX 11: LibLINEAR

APPENDIX 12: Logistic Regression

APPENDIX 13: Bagging

APPENDIX 14: Boosting

APPENDIX 15: Artificial Neural Networks

APPENDIX 16: Nearest Neighbors

APPENDIX 17: RapidMiner

LIST OF TABLES

	Page
Table 3.1. Details of dataset.....	8
Table 3.2. Details of software metrics.....	10
Table 4.1. Default and improved configurations of Decision Tree in WEKA.....	14
Table 4.2. Performance results of default and improved configurations of Decision Tree in WEKA.....	15
Table 4.3. Performance results of Subtree Raising and Unpruned configurations of Decision Tree in WEKA.....	15
Table 4.4. Confusion matrix of default configurations of Decision Tree in WEKA.....	15
Table 4.5. Confusion matrix of improved configurations of Decision Tree in RapidMiner.....	16
Table 4.6. Default and improved leaf size count of Decision Tree in RapidMiner.....	16
Table 4.7. Performance results of leaf size count of Decision Tree in RapidMiner.....	17
Table 4.8. Confusion matrix of 2 leaf size of Decision Tree in RapidMiner.....	17
Table 4.9. Confusion matrix of 5 leaf size of Decision Tree in RapidMiner.....	18
Table 4.10. Default and improved iterations count of RF in WEKA.....	19
Table 4.11. Performance results of iterations count of RF in WEKA.....	19
Table 4.12. Performance results of maximum depth of tree count of RF in WEKA.....	19
Table 4.13. Confusion matrix of 100 iteration of RF in WEKA.....	20
Table 4.14. Confusion matrix of 500 iteration of RF in WEKA.....	20
Table 4.15. Criterion performance results of RF in RapidMiner.....	21
Table 4.16. Confusion matrix of Gain Ratio of RF in RapidMiner.....	22
Table 4.17. Confusion matrix of Information Gain of RF in RapidMiner.....	22
Table 4.18. Performance results of search algorithms.....	24
Table 4.19. Confusion matrix of K2 search algorithm with 2 parent number of Bayesian Network.....	25

Table 4.20. Confusion matrix of K2 search algorithm with 3 parent number of Bayesian Network.....	25
Table 4.21. Kernel Estimator congifuration for improving Naïve Bayes performance in WEKA.....	27
Table 4.22. Performance results of kernel estimators of Naïve Bayes in WEKA.....	28
Table 4.23. Confusion matrix of without kernel estimator of Naïve Bayes in WEKA....	28
Table 4.24. Confusion matrix of kernel estimator of Naïve Baye in WEKA.....	28
Table 4.25. Performance results of Naïve Bayes in RapidMiner.....	29
Table 4.26. Confusion matrix of Naïve Bayes in RapidMiner.....	29
Table 4.27. Confusion matrix of Naïve Bayes Kernel in RapidMiner.....	30
Table 4.28. Performance results of default and improved configurations of Rule Based classification.....	31
Table 4.29. Confusion matrix of 2 minimum number of instance with 0.25 confidence factor of Rule Based classification.....	31
Table 4.30. Confusion matrix of 4 minimum number of instance with 0.25 confidence factor of Rule Based classification.....	32
Table 4.31. Kernel functions formulas and parameters.....	33
Table 4.32. Performance results of kernel types of SMO in WEKA.....	34
Table 4.33. Confusion matrix of Polynomial kernel of SMO in WEKA.....	35
Table 4.34. Confusion matrix of PUK kernel of SMO in WEKA.....	35
Table 4.35. Performance results of kernel types of SMO in RapidMiner.....	36
Table 4.36. Confusion matrix of Polynomial kernel of SMO in RapidMiner.....	36
Table 4.37. Confusion matrix of Dot kernel of SMO in RapidMiner.....	36
Table 4.38. Performance results of kernel types of LibSVM in WEKA.....	38
Table 4.39. Performance results of Radial Basis of LibSVM in WEKA.....	38
Table 4.40. Performance results of Linear kernel of LibSVM in WEKA.....	38
Table 4.41. Performance results of kernel types of LibSVM in RapidMiner.....	39
Table 4.42. Confusion matrix of Radial Basis kernel of LibSVM in RapidMiner.....	40

Table 4.43. Performance results of SVM types of LibLINEAR in WEKA.....	42
Table 4.44. Confusion matrix of L2-Regularized L2-Loss Support Vector Classification (Dual) of LibLINEAR in WEKA.....	43
Table 4.45. Confusion matrix of L1-Regularized Logistic Regression of LibLINEAR in WEKA.....	43
Table 4.46. Confusion matrix of LibLINEAR default configurations in RapidMiner...	44
Table 4.47. Confusion matrix of default configurations of LR in WEKA.....	45
Table 4.48. Confusion matrix of default configurations of LR in RapidMiner.....	46
Table 4.49. Performance results of Bagging classifiers in WEKA.....	47
Table 4.50. Confusion matrix of REPTree in Bagging for WEKA.....	48
Table 4.51. Confusion matrix of RF in Bagging for WEKA.....	48
Table 4.52. Performance results of Bagging classifiers in RapidMiner.....	49
Table 4.53. Confusion matrix of Decision Tree classifier in Bagging for RapidMiner...	50
Table 4.54. Confusion matrix of RF classifier in Bagging for RapidMiner.....	50
Table 4.55. Performance results of Boosting classifiers in WEKA.....	51
Table 4.56. Confusion matrix of Decision Stump in Boosting for WEKA.....	52
Table 4.57. Confusion matrix of RF in Boosting for WEKA.....	52
Table 4.58. Performance results of Boosting classifiers in RapidMiner.....	53
Table 4.59. Confusion matrix of Decision Stump classifier in Boosting for RapidMiner	54
Table 4.60. Confusion matrix of RF classifier in Boosting for RapidMiner.....	54
Table 4.61. Performance results of Training Cycle, Learning Rate, Momentum combinations of ANN in WEKA.....	57
Table 4.62. Confusion matrix of 0.3 learning rate, 500 training cycles and 0.2 momentum combination of ANN in WEKA.....	58
Table 4.63. Confusion matrix of 0.01 learning rate, 500 training cycles and 0.2 momentum combination of ANN in WEKA.....	58
Table 4.64. Performance results of Training Cycle, Learning Rate, Momentum combinations of ANN in RapidMiner.....	59

Table 4.65. Confusion matrix of 0.3 learning rate, 500 training cycles and 0.2 momentum combination of ANN in RapidMiner.....	60
Table 4.66. Confusion matrix of 0.01 learning rate, 500 training cycles and 0.9 momentum combination of ANN in RapidMiner.....	60
Table 4.67. Distance functions formulas and parameters.....	62
Table 4.68. Performance results of search algorithms and distance functions of KNN in WEKA.....	63
Table 4.69. Confusion matrix of Linear search with Euclidean distance of KNN in WEKA.....	64
Table 4.70. Confusion matrix of Linear search with Manhattan distance of KNN in WEKA.....	64
Table 4.71. Performance results of search algorithms and distance functions of KNN in RapidMiner.....	65
Table 4.72. Confusion matrix of Euclidean distance of KNN in RapidMiner.....	65
Table 4.73. Confusion matrix of Manhattan distance of KNN in RapidMiner.....	65
Table 5.1. Faulty and not faulty class confusion matrix	67
Table 5.2. Performance results of reliability prediction by default values of WEKA.....	69
Table 5.3. Performance results of reliability prediction by default values of RapidMiner..	70
Table 5.4. Performance results of reliability prediction by improved values of WEKA.....	71
Table 5.5. Performance results of reliability prediction by improved values of RapidMiner	72
Table 5.6. AUC results of WEKA and RapidMiner by improved values.....	73

LIST OF FIGURES

	Page
Figure 4.1. ROC curves of default and improved configurations of Decision Tree in WEKA.....	16
Figure 4.2. ROC curve of 5 leaf size of Decision Tree in RapidMiner.....	18
Figure 4.3. ROC curves of 100 and 500 iteration of RF in WEKA.....	21
Figure 4.4. ROC curve of Information Gain of RF in RapidMiner	22
Figure 4.5. ROC curves of K2 search algorithm with 2 and 3 parent number of Bayesian Network.....	24
Figure 4.6. Bayesian Network formed for 9 software prediction metrics.....	26
Figure 4.7. Bayesian Network formed for 20 software prediction metrics.....	26
Figure 4.8. ROC curves of kernel estimators of Naïve Bayes in WEKA.....	29
Figure 4.9. ROC curve of Naïve Bayes Kernel in RapidMiner.....	30
Figure 4.10. ROC curves of 2 and 4 minimum number of instance with 0.25 confidence factor of Rule Based classification.....	32
Figure 4.11. ROC curves of Polynomial and PUK kernels of SMO in WEKA.....	35
Figure 4.12. ROC curve of Dot kernel of SMO in RapidMiner.....	37
Figure 4.13. ROC curves of Radial Basis and Linear kernels of LibSVM in WEKA...	39
Figure 4.14. ROC curve of Radial Basis kernel of SMO in RapidMiner.....	40
Figure 4.15. ROC curves of L2-Regularized L2-Loss Support Vector Classification (Dual) and L1-Regularized Logistic Regression of LibLINEAR in WEKA.....	43
Figure 4.16. ROC curve of LibLINEAR default configurations in RapidMiner.....	44
Figure 4.17. ROC curve of default configurations of LR in WEKA.....	45
Figure 4.18. ROC curve of default configurations of LR in RapidMiner.....	46
Figure 4.19. ROC curves of REPTree and RF in Bagging for WEKA.....	48
Figure 4.20. ROC curve of RF in Bagging for RapidMiner.....	50
Figure 4.21. ROC curves of Decision Stump and RF in Boosting for WEKA.....	52

Figure 4.22. ROC curve of RF in Boosting for RapidMiner.....	54
Figure 4.23. Neural Network graphical user interface form for 20 software metrics....	56
Figure 4.24. Neural Network graphical user interface form for 9 software metrics.....	56
Figure 4.25. ROC curves of 0.3 learning rate, 500 training cycles, 0.2 momentum and 0.01 learning rate, 500 training cycles, 0.2 momentum combinations of ANN in WEKA.....	59
Figure 4.26. ROC curves of 0.01 learning rate, 500 training cycles, 0.9 momentum of ANN in RapidMiner.....	61
Figure 4.27. ROC curves of Euclidean with Manhattan distance of KNN in WEKA...	64
Figure 4.28. ROC curve of Manhattan distance of KNN in RapidMiner.....	66
Figure 5.1. Default configuration of SVMs ROC curves.....	74
Figure 5.2. Improved configuration of SVMs ROC curves.....	75
Figure 5.3. Default configuration of Bagging and Boosting ROC curves.....	76
Figure 5.4. Improved configuration of Bagging and Boosting ROC curves.....	77
Figure 5.5. Improved configuration of RF and Naïve Bayes ROC curves.....	77
Figure 5.6. Improved configuration of RF and Nearest Neighbors ROC curves.....	78
Figure 5.7. Improved configuration of RF and Decision Tree ROC curves.....	79

LIST OF SYMBOLS AND ABBREVIATIONS

AMC	Average Method Complexity
ANN	Artificial Neural Network
AUC	Area Under the Curve
AVG_CC	Mean Values of Methods
CA	Afferent Couplings
CAM	Cohesion Among Methods of a Class
CBM	Coupling Between Methods
CBO	Coupling Between Objects
CE	Efferent Couplings
CK	Chidamber and Kemerer
DAM	Data Access Metric
DIT	Depth of Inheritance Tree
FN	False Negative
FP	False Positive
GRNN	General Regression Neural Network
IC	Inheritance Coupling
KNN	K-Nearest Neighbors
LCOM	Lack of cohesion of methods
LCOM3	Lack of Cohesion Among Methods of a Class 3
LMT	Logistic model tree
LOC	Line of Code
LOCQ	Quality of Source Code
LR	Logistic Regression
MAE	Mean Absolute Error
MAX_CC	Maximum Values of Methods
MFA	Measure of Functional Abstarction
ML	Machine Learning
MLP	Multilayer Perceptron
MOA	Measure of Aggression
MPC	Message Passing Coupling
NB	Naïve Bayes
NOC	Number of Children
NOM	Number of Methods
NPM	Number of Public Methods
OO	Object-Oriented
PUK	Pearson VII kernel function
RF	Random Forest
RFC	Response for a Class
RMSE	Root Mean Square Error
ROC	Receiver Operating Characteristics
SLOC	Source Lines of Code
SMO	Sequential Minimal Optimization
SVM	Support Vector Machines
TN	True Negative
TP	True Positive
WEKA	Waikato Environment for Knowledge Analysis
WMC	Weighted Methods per Class

1 INTRODUCTION

Software quality and reliability are the most crucial aspects related to software complexity. In software engineering, software defects prediction with reliability is a problematic area. Reliability measures the number of bugs in the developed software systems. The software complexity level is also associated with maintainability. Software defect prediction research area is crucial for the reliability and it provides predicting defective or fault-prone modules in software. As a result, software projects should be reliable and cost-effective. The OO method is different from the conventional programming strategy. It distributes data and controls the objects. The OO method has become significant in software engineering than structural design. It demonstrates the new potential and more reliable way to analyze the problems. The OO method provides better reusability, reliability and maintainability than the traditional approach. The software metrics can be used to evaluate the quality of OO software. OO software and these metrics support to confirm the quality properties of software such as fault proneness.

1.1 Problem Definition

Software requirements, needs, complexities and modules are growing day by day because of the large software systems. The most critical issues for the customer are the usability and reliability of software for product satisfaction. A bug in a software project can also be called a defect. Finding and eliminating software defects are one of the most significant problems. This seems to be a common problem in the software development life cycle. It is a challenging task because the failure rate potential may become increase when software complexity level increases and it affects product satisfaction. Nevertheless, the defect-free software product is almost impossible. Thus, the prediction of faulty classes is difficult to handle because of the increasing software line of code. One way to overcome these problems is a traditional process such as code review, software testing like unit, integration and system testing. Nevertheless, the effort, time, cost and resource increase because of the growth of software complexity. Complex software projects require a significant and costly testing process. In previous studies, software metrics and techniques are reported to be good indicators of software reliability.

1.2 Problem Solution

Most of the research in this field aimed at solving the minimization of defects, determining fault-prone software classes with predictive models, code changes and software metrics. The most useful way is to predict or detect faulty classes or modules in the early stage of the software development life cycle. Consequently, the quality of software products may improve, help reliable software, reduce testing and maintenance costs, allocate testing resources efficiently, support software testing activities and optimize engineer efforts before the software is released. Useful software metrics, especially OO can determine software quality and performance. OO paradigm provides a better design solution and reliability issue than the traditional approach. There are many ML algorithms in the literature that might predict the reliability of software. The main objective is to investigate comparative performance analysis with a 10-fold cross-validation technique performed to find the accuracy of the training model between different 11 most widely used ML techniques and two libraries for SVM. These techniques are Decision Tree, Rule-based Classification, SVM, Nearest Neighbors, Bayesian Network, Random Forest (RF), Logistic Regression (LR), Boosting, Bagging, Naïve Bayes, Neural Networks. SVM library's names are Library for Support Vector Machines (LibSVM) and Library for Large Linear Classification (LibLinear). Moreover, combinations of various kernel types, classifiers, estimators, measure types, iterations and search algorithms were analyzed for improving prediction accuracy. Data set includes 33 open sources OO software and 8093 instances without noisy data obtained from the PROMISE repository and 20 software metrics for determining the best technique and effect of OO metrics, specifically CK metrics suite on defect prone classes in order to the generalized conclusion.

These ML techniques evaluated six evaluation metrics such as Accuracy, Precision, Recall, AUC, MAE and RMSE. Besides, the ROC used for solving the dataset imbalanced nature problem. These performance metrics were calculated with RapidMiner and WEKA ML tools. In this way, the performance of both tools compared. Moreover, there is no previous research analyzes the same dataset with both tools. This approach can solve another performance comparison problem between WEKA and RapidMiner tools because both tools calculated performance slightly differently.

Various metrics have been proposed in the literature by researchers to evaluate the quality of software. Nevertheless, CK metrics good indicator of reliability and maintainability issues [48].

1.3 Study Motivation

Previous research and literature reviews show that various ML techniques such as SVM, Bayesian learning, Decision Trees, Neural Networks, ensemble and Rule-based learning models were used. In previous studies several software metrics were proposed to find or develop efficient defect prediction techniques. However, limited data sets and software metrics were analyzed. There are no comprehensive analyses evaluating ML algorithms with a large dataset and using two different ML tools for software reliability. Accordingly, the critical problem with these analyses is that different ML performance results available in the literature. Another constraint and common problem can be reported as the limited software metrics are available in the dataset.

1.4 Aims & Objectives

Research aims at finding a solution to this challenging problem of software reliability issues and how some constraints such as time and cost-efficiently should be used. These constraints make the problem difficult. Various ML techniques have been used to overcome this problem and detect reliability issues. These techniques are effective technologies. Useful information can be retrieved from these techniques for developers and managers. Consequently, resource allocation can be more stable, coding and design quality improve, testing cost and maintenance effort decrease. Another important goal is to improve the classification accuracy of the ML algorithms. Thus, these techniques can be used in bug prediction, effort estimation and risk judgment in the early phases of the software development life cycle. Also, it can be useful for minimization of business risk.

In the first part of the thesis, primary problem definition and solution, study motivation, objectives proposal summarized. In Section 2, an overview of previous studies presented about software reliability and related topics. Moreover, the limitations of previous research and research questions are given. In Section 3, 20 software metrics, their usage and 33 datasets are explained. In Section 4, the applied ML techniques, used ML tools such as WEKA and RapidMiner have described. In Section 5, summarizes the main findings, strengths and weaknesses of methods and performance evaluation results such as Accuracy, AUC, Precision, Recall, MAE, RMSE and ROC. Furthermore, a comparison of each technique discussed. Proposed ML techniques for defect prediction also discussed in this section. As a result, conclusions and future works given in Section 6.

2 LITERATURE REVIEW

2.1 Previous Literature

Previous literature shows that different regression models, ML techniques and software metrics have been proposed to determine and predict defect-prone classes or modules for software quality prediction. There have been different efforts in this area. In this section, previous related works using different ML techniques with various software metrics presented.

Several studies suggest that CK software design level metrics are used to determine software quality. This metrics suite has indicated that the prediction of defects efficiently and it can be a good indicator of software quality. CK metrics suite included six essential metrics. These are WMC, CBO, RFC, NOC, LCOM, DIT. These OO software metrics can measure the reusability, reliability, maintainability of software and point of the class hierarchy [47].

Basili et al. research has emphasized OO software metrics impact of fault prediction in the early stages of the development life cycle. In this Basili study, Multivariate logistic regression is selected in prediction techniques for proving the effect of OO software metrics [53].

Several OO software metrics have been proposed, such as Number of Methods (NOM), Message Passing Coupling (MPC), Size of Procedures or Functions (SIZE1), Number of Methods (NOM), Data Abstracting Coupling (DAC), Properties Size defined in a class (SIZE2) assessed the maintenance effort and relation between of each metric in Li & Henry research. Li & Henry recommended these software metrics to measure the number of changes, complexity and coupling levels in the software using these OO metrics [55].

Numerous studies to investigate the primary attribute of software metrics for software quality. Tang et al. reported that the WMC metric is a significant predictor for faulty modules and classes [39]. Another research is Briand et al., which extracted 49 software metrics proposed using Multivariate Logistic Regression models. The result of the research shows that all metrics can be a good predictor, except NOC [32].

Quah & Thwin research analyzed different Neural Networks for estimating maintainability and reliability of software quality such as General Regression Neural Network (GRNN) and Ward Neural Network using OO software metrics. After both Neural

Networks compared with each other, the GRNN model has the best performance for maintainability and reliability [40].

Gyimothy et al. research suggests that the CBO metric is the primary indicator and LOC performance is also useful for fault prediction. Nevertheless, the NOC attribute have low results in the CK metrics suite. In this research, the Mozilla dataset used in Decision Trees, LR and Neural Network [50].

Furthermore, different software metrics were analyzed except the CK metrics suite. For example, Zimmermann et al. used 14 size metrics in Eclipse Java source code and tried to predict fault and non-faulty classes. Efficiently predict faults, a combination of complexity metrics suggested [52].

Nasa KC1 is the most analyzed dataset for fault-prone prediction. This dataset was analyzed with the Decision Tree and Neural Network with ROC analysis to predict fault proneness and validated with high, medium and low defects severity levels. Based on the Decision Tree and Neural Network performance results, both techniques are better than the LR [57]. Another research considered the effectiveness of the prediction of fault-prone classes and OO software metrics [42]. Studies of Pai and Dugan evaluated the Nasa KC1 dataset also with using Bayesian Network. Bayesian Network form result shows that WMC, RFC, CBO and Source Lines of Code (SLOC) metrics are the major contributor for fault proneness of software prediction [43]. Elish et al. 's findings suggest that the SVM has the same NASA dataset and purpose. SVM has better performance than the other eight compared to ML techniques [27]. Other experiments and alternatives ML techniques analyzed for reliability [24, 25].

Some studies such as Arisholm et al. suggested the AdaBoost algorithm. LR, SVM and Neural Networks techniques were compared to different options of Decision Tree with Java telecom system. If the AdaBoost technique used with C4.5, performance gave a useful result [6].

Malhotra, Sawhney and Shukla analyzed the performances of several 17 ML methods on Xerces OO software to find defect prone classes. Bagging method is the most reliable performance for defect prediction models [41].

Unlike traditional software metrics, Okutan & Yildiz research evaluated two different software metrics associated with the number of developers (NOD) and the quality of source code (LOCQ). Bayesian Networks used to observe the relation between defect proneness and new software metrics. This work shows that RFC, LOC and LOCQ are more remarkable

attributes, whereas LCOM, WMC and CBO software metrics are less useful on software quality prediction. Another conclusion is that NOC and DIT software metrics are not a good indicator [2].

Over time, an extensive literature has investigated software reliability using PROMISE data repository and maintainability using QUES systems. ML techniques are not particularly new and have been used for many years in software reliability. For instance, Reddivari and Raman compared eight ML techniques. RF's performance results in terms of AUC value is a good predictor for maintainability and reliability. When we consider software CK metrics suite, Coupling Between Methods (CBM), WMC and DIT attributes are important for software quality [48].

2.2 Limitations of Previous Research

Previous studies have shown that different ML techniques and software metrics were analyzed and proposed. However, several gaps and limitations are still available. In previous studies, the datasets to investigate software reliability were limited. As far as we know, less previous research has investigated large datasets and the effect of CK software metrics on software reliability. However, the existing research has some problems with the generalization of results, since performance results may change according to dataset size and characteristics. In the past several researches, the various dataset included different software metrics suite and characteristics of datasets are not same. For example, in this thesis, all datasets are OO software and include CK metrics attributes. To fill this literature gap, this thesis has large dataset that included 20 software metrics.

2.3 Research Questions

The literature review shows that OO software metrics, especially CK metrics suite a crucial predictor of quality rather than the other traditional software metrics. In this study, following research questions are set as below. To the best of our knowledge, these questions have previously never been addressed.

RQ1. What is the most and least effective software metric to determine software reliability?

RQ2. What are the most effective ML techniques to estimate software defect prediction?

RQ3. Is it possible to generalize the best performance result of the ML techniques to estimate software defect prediction?

RQ4. Is the performance results differ according to Machine Learning tool used?

3 MATERIALS & METHODS

3.1 Data Collection

In this thesis study, 33 open-source of datasets received from software projects. The latest versions of projects were selected because duplicate classes and values should prevent. This dataset includes 8093 instances and 2958 defects. The software defect prediction public dataset obtained from one of the most popular and largest data repositories called PROMISE [17]. Names of project dataset are Ant, Tomcat, Velocity, Ivy, JEdit, Poi, Forrest, Workflow, Log4j, Lucane, Synapse, Xalan, Arc, Berek, Camel, CKJM, E-Learning, Intercafe, Kalkulator, Nieruchomosci, Pbeans, Pdftranslator, Prop, Redaktor, Serapion, Skarbonka, Sklebagd, System data, Szybkafucha, Termoproject, Wspomaganiepi, Xerces, Zuzel. All these datasets contain the same OO software metrics. Moreover, all dataset gives information about the bug at the class level. In Table 3.1, the dataset is given in detail. Xalan and Log4j software have the highest defective modules. JEdit and Forrest software has the least defective modules. However, Xalan and Tomcat software have the most instances. CKJM and Wspomaganiepi dataset have the least instances. All dataset includes CK metrics. For analyses of these datasets in WEKA and RapidMiner, all dataset combined in one CSV file without software name and version attributes. Because the name and version of the software should not affect the target class determination and performance results. For this purpose, the version and name of the software was not selected as an attribute in the RapidMiner ML tool. The analyzed data does not include missing data. However, bug label in all dataset contains a count of bug. For ML techniques, these bug counts change to binary classification as 1 is defective, 0 is a non-defective software module.

Table 3.1 Details of dataset

Dataset	Release	# of Instance	# Defective Module	% Defective Module	Metrics Suite
Ant	1.7	745	166	22	CK
ArcPlatform	1.0	234	27	12	CK
Berek	1.0	43	16	37	CK
Camel	1.6	965	188	19	CK
CKJM	1.8	10	5	50	CK
E-Learning	1.0	64	5	8	CK
Forrest	0.8	32	2	6	CK
Intercafe	1.0	27	4	15	CK
Ivy	2.0	352	40	11	CK
JEdit	4.3	492	11	2	CK
Kalkulator	1.0	27	6	22	CK
Log4j	1.2	205	189	92	CK
Lucene	2.4	340	203	60	CK
Nieruchomosci	1.0	27	10	37	CK
Pbeans	2.0	51	10	20	CK
Pdftranslator	1.0	33	15	45	CK
Poi	3.0	442	281	64	CK
Prop	452	660	66	10	CK
Redaktor	1.0	176	27	15	CK
Serapion	1.0	45	9	20	CK
Skarbonka	1.0	45	9	20	CK
Sklebagd	1.0	20	12	60	CK
Synapse	1.2	256	86	34	CK
Systemdata	1.0	65	9	14	CK
Szybkafucha	1.0	25	14	56	CK
Termoproject	1.0	42	13	31	CK
Tomcat	6.0	858	77	9	CK
Velocity	1.6	229	78	34	CK
Workflow	1.0	39	20	51	CK
Wspomaganiepi	1.0	18	12	67	CK
Xalan	2.7	909	898	98	CK
Xerces	1.4	588	437	74	CK
Zuzel	1.0	29	13	45	CK

3.2 Software Metrics

Software metrics revealed a significant role for measuring software characteristics and complexity level. The metrics suite aims to figure out various problems, such as software maintainability and reliability. Previous research shows that a lot of traditional and OO software metrics suite has been suggested. Mostly used software metrics suites are Bansiya and Davis [7], Briand et al. [32], CK metrics suite [47], Etzkorn et al. [34], Genero et al. [19], Halstead [36], Harrison et al. [22], Kim and Ching [30], Li and Henry [55], Lorenz and Kidd [38], McCabe's cyclomatic complexity [51], Melo and Abreu [56], MOOD Metrics [16], Robert Martin [44], Sharble and Cohen [49], Tang, Kao and Chen [39], Tegarden et al. [12]. In thesis study used the CK software metrics suite because one of the most important and widely used OO metrics suite. Evaluating of external software quality attributes such as maintainability, reliability and reusability, CK metrics suite widely used in the past research. Besides, internal characteristics of software systems like class complexity, cohesion, coupling, encapsulation, inheritance and polymorphism can be measured by software metrics. Details of 20 software metrics is given in Table 3.2. This research intends to understand the relationship between reliability and CK metrics suite. The literature review and previous research show that the CK metrics suite has been provided significantly effective for predicting faults in a software system.

Table 3.2 Details of software metrics

Metric Name	Description
Afferent Couplings (CA)	It is Martin's Metric. It is the count of classes which is calling given or a particular class.
Average Method Complexity (AMC)	AMC is the count of binary codes, which means the average McCabe's Cyclomatic Complexity method size.
Average McCabe's Cyclomatic Complexity (AVG_CC)	Average count of the independent values of methods.
Cohesion Among Methods of a Class (CAM)	It is a similarity method related to signatures and prototypes.
Coupling between Methods (CBM)	Measure the total count of new or redefined methods to which are inherited methods that are coupled [40].
Coupling Between Objects (CBO)	Total count of classes to couple with or given class. Maintainability can be difficult when the number of coupling number is high. The level should be low to maintainability.
Data Access Metric (DAM)	The ratios of all private and non-public (protected) attributes are divided by all class attributes. It is related to encapsulation.
Depth of Inheritance Tree (DIT)	It is shown that the maximum level. In other word, longest path from the class to the root of the tree [47]. If a high count of steps from the root to the leaf node or inherited methods increase software complexities. So, the level should be low.
Efferent Couplings (CE)	It is Martin's Metric. It is a count of classes, which is called a given or specific class.
Inheritance Coupling (IC)	It is a count of parent classes to couples with a given or specific class.
Lack of Cohesion of Methods (LCOM)	The number of methods pairs not having any common attributes that pass references to variables. High LCOM shows that weak encapsulation is available in the software. It should be low.
Lack of Cohesion Among Methods of a Class 3 (LCOM3)	It is minor improvements and variation of cohesion level calculation of the LCOM by Henderson-Sellars.
Line of Code (LOC)	In the method, the number of code lines with non-commented. Comment and blank lines do not include.
Maximum McCabe's Cyclomatic Complexity (MAX_CC)	It is the maximum count of the independent values of the method.
Measure of Aggression (MOA)	It is the percentage of user-defined data declarations in the given class. It is associated with the composition.
Measure of Functional Abstraction (MFA)	It is the ratio in the count of methods. These methods inherited by a class divided by the total defined methods number available [53]. It is associated with inheritance.
Number of Children (NOC)	NOC is associated with inheritance. Because it measures the count of subclasses or direct children of a given class. It should be low to maintainability of the software.
Number of Public Methods (NPM)	It is the count of convenient public methods in a given class.
Response for a Class (RFC)	It responds to a message obtained by some method or class object after a set of all methods in a class executed [47]. Internal and external methods in a class. It should be low.
Weighted Methods Per Class (WMC)	It is a CK metric. Sum of methods complexities in a class. High WMC values indicate a more complex software class. If software complexity increases, software class maintainability can be more difficult. [47, 40]. The complexity level should be low.
Defect Count	It is binary classification shown as absence or presences of the defects.

3.3 Machine Learning Software Tools

ML software is a data mining tool that includes various ML algorithms. Tools contain regression, data process, classification, clustering, data mining and visualization features. For using ML algorithms, most popular tools RapidMiner and WEKA are selected. Both tools are easy to use, fast, stable and provide various ML algorithms. Moreover, both tools provide visualization and different performance analyses features. ML tools provide various features. For instance, train models, enable them to find new methods and support deep learning.

3.3.1 WEKA machine learning software

Weka is an open-source software tool. It can use it through a graphical user interface. Weka images interface available in appendix 1.

- Preprocess section provide edit and add features of dataset columns and display statistic of selected attribute. Also, the preprocess section allows various modifications of columns. For example, in dataset Table 3.1, all attributes are numeric. Defect attributes should be binary value mean “True” or “False”. For this reason, defect attributes changed to the “NumericToNominal” below “unsupervised” filter. Interfaces available in appendix 2.
- Classify sections provide various ML algorithms. It provides an analysis of the dataset. Furthermore, different test options like cross-validation, training and test set can be prepared.
- Select attributes section enables to find and analysis of major and minor attributes in the dataset.

Weka Knowledge Flow environment also used to compare ROC curves. Weka Knowledge Flow interface available in appendix 3.

- Arrf loader import to arrf file format.
- Numeric to nominal section change to numeric attributes into nominal.
- Class assigner set or unset to the configuration of the class index.
- Cross-validation fold maker provides number of folds and random seed configurations.
- The classifier of the performance evaluator section manages to evaluate metrics.
- The performance chart of the model visualizes the performance metrics such as ROC curve.

3.3.2 RapidMiner machine learning software

RapidMiner can work on cross-platform operating systems and has a friendly user interface. It enables to create own models with drag and drop feature. It provides data preparation, text mining and deep learning. In the RapidMiner IDE, a visual representation of the data mining process flow is available [46].

- The data access section enables us to read and write a file with different formats such as CSV, Excel and ARFF. Also, it provides read and writes from database and cloud storage.
- The blending section provides a modification of columns, transformation attributes to different types and join features.
- The modeling section provides various ML algorithms, correlations matrix, feature weights and optimizations.
- The validation section provides various predictive performance features and visualizations.

RapidMiner interface available in appendix 17.

4 TECHNIQUES & EXPERIMENTS

ML algorithms are well proven to be significantly effective in software maintainability and reliability. ML prediction and classification techniques provide useful information from the dataset for software quality. In this chapter, mostly used ML techniques were analyzed with different parameters and options in WEKA and RapidMiner tools. Thus, the effectiveness of techniques can be evaluated for dataset. Default and different configuration of each ML techniques compared to each other for improving performance results in WEKA and RapidMiner tools. In used dataset, the bug label should be changed numeric to a nominal attribute.

Furthermore, Decision Stump, Random Tree, Logistic Model Tree (LMT) classifiers analyzed in Bagging and Boosting algorithms. Decision Stump is a one-level simple tree structure of the Decision Tree model. Also, it is called 1-rules, which uses a single instance for splitting. Random Tree can deal with both regression and classification problems. Multiple Random Trees consist of the RF method. LMT is logistic model tree for classification problems. It consists of classification trees with a logistic regression model and decision tree at the leaves.

Moreover, various criteria analyzed, such as information gain, gain ratio and gini index. All the features or attribute entropies calculated to find the best feature and maximum information about a class for information gain. Try to decrease the level of entropy, which is starting from the root to the leaf nodes. It is used by ID3 and C4.5 algorithms. The gain ratio is an alternative of information gain that modifies the information gain. Gini index calculates the amount of specific feature probability. These specific features mislabeled. It is also called gini impurity and used by CART algorithms.

Others affect the pruning rule is the confidence factor parameter and reduced error pruning. For little pruning, confidence factor value should be large. For heavy pruning, confidence factor should be small. The confidence factor threshold range should be between 0 and 1 and shows that internal error while pruning the decision tree in data. If a large amount of data is analyzed, reduced error pruning can be useful. Because decision tree branches may have noise data. These branches can be removed and identified. As a result, the classification result improves. Nevertheless, scalability can be a problem if a large amount of dataset is

available [45]. It is starting and checking each internal node for changing it with the most popular or repeated class that does not decrease the prediction accuracy of the trees. [3].

4.1 Decision Tree

The decision tree is a classification and predictive model. It is flow-chart and tree-classifying the instance by sorting them down the tree where each node shows all reachable decisions with edges. Decision tree algorithms such as C4.5 and ID3 are successful algorithms and try to find an attribute that returns the highest information gain. The decision tree is available as J48 in Weka [37]. Screenshot of WEKA tool usage is available in appendix 4 for Decision Tree.

The confidence factor default value is 0.25 in Weka. For comparison between RapidMiner and WEKA, RapidMiner confidence factor specified 0.25 and there were no additional filters in both tools in Table 5.2 and Table 5.3. Value of the improved attributes to achieve higher AUC performance is given in Table 4.1 for WEKA and is given in Table 4.6 for RapidMiner.

4.1.1 Decision Tree analyses in WEKA

Table 4.1 Default and improved configurations of Decision Tree in WEKA

Settings	Default Value	Improvement
Confidence Factor	0.25	0.1
Reduced Error Pruning	False	True

The default value of the “confidence factor” parameter is 0.25, the “reduced error pruning” parameter is false. After a series of experiments, the “confidence factor” should be 0.25 and “reduced error pruning” should be true to get a better performance result. The differences between the performance results of default and improved values are given in Table 4.2.

Table 4.2 Performance results of default and improved configurations of Decision Tree in WEKA

Confidence Factor	Reduced Error Pruning	Accuracy	AUC	Precision	Recall	MAE	RMSE
0.25 (Default)	False (Default)	72.25%	0.694	0.715	0.722	0.325	0.480
0.1	False	73.15%	0.703	0.725	0.731	0.336	0.460
0.25 (Improved)	True (Improved)	73.13%	0.724	0.725	0.731	0.340	0.453

Only reduced error pruning should be true to get the highest AUC value for Decision Tree. The default value of “subtreeRaising” is true and “unpruned” is false. Subtree raising is post-pruning operations. It replaces or reclassify a tree with one of its subtrees and has a minor effect on the decision tree. The unpruned decision trees are larger than pruned trees. If the tree is pruned, the size of branches or nodes can be removed. If “subTreeRaising” options set to false or “unpruned” set to true, the value of AUC and accuracy results decrease. Performance results of “subTreeRaising” and “unpruned” configuration is given in Table 4.3.

Table 4.3 Performance results of Subtree Raising and Unpruned configurations of Decision Tree in WEKA

Subtree Raising	Unpruned	Accuracy	AUC	Precision	Recall	MAE	RMSE
False	True	71.46%	0.698	0.709	0.715	0.315	0.494

Confusion matrix of 0.25 confidence factor without reduced error pruning is given in Table 4.4. Confusion matrix of 0.25 confidence factor with reduced error pruning is given in Table 4.5. Comparison of default and improved ROC curves are given in Figure 4.1.

Table 4.4 Confusion matrix of default configurations of Decision Tree in WEKA

Classified As	a	b
a = 0	4273	862
b = 1	1384	1574

Data are classified 72% correctly. According to the confusion matrix, 862 instances should be non-defect but classified as defective and 1384 instances should be defective but classified as non-defective.

Table 4.5 Confusion matrix of improved configurations of Decision Tree in WEKA

Classified As	a	b
a = 0	4456	679
b = 1	1496	1462

Data are classified 73% correctly. According to the confusion matrix, 679 instances should be non-defect but classified as defective and 1496 instances should be defective but classified as non-defective.

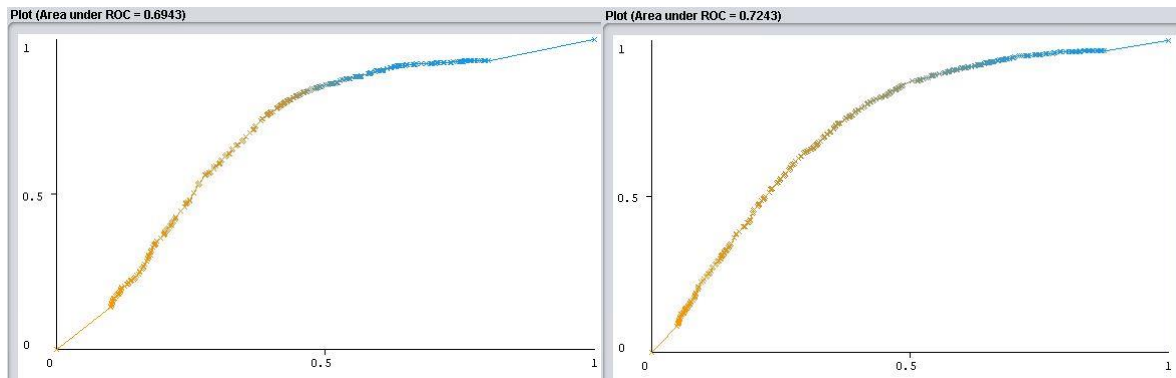


Figure 4.1 ROC curves of default and improved configurations of Decision Tree in WEKA

4.1.2 Decision Tree analyses in RapidMiner

For default and improved RapidMiner configuration, the confidence factor should be 0.25 to comparing with the WEKA tool.

Table 4.6 Default and improved leaf size count of Decision Tree in RapidMiner

Settings	Default Value	Improvement
Minimal Leaf Size	2	5

The default value of the minimal leaf size is 2. After a series of experiments, the minimal leaf size should be 5 to get a better performance result. Leaf size is the number of observations or cases in that leaf.

The end of the node is a leaf in a decision. Minimal leaf size use for a more prone model to find noise in train data. The differences between the performance results of default and improved leaf size are given in Table 4.7.

Table 4.7 Performance results of leaf size count of Decision Tree in RapidMiner

Minimal Leaf Size	Pre-Pruning	Accuracy	AUC	Precision	Recall	MAE	RMSE
2 (Default)	TRUE (Default)	72.94%	0.730	0.738	0.886	0.346	0.440
3	TRUE	73.04%	0.733	0.739	0.888	0.347	0.439
4	TRUE	72.74%	0.732	0.737	0.885	0.349	0.439
5 (Improved)	TRUE	72.82%	0.735	0.737	0.887	0.351	0.437
6	TRUE	72.77%	0.734	0.737	0.887	0.351	0.437

In this thesis study, the experiment results show that minimal leaf size should be 5 to get the highest AUC value for Decision Tree. Nevertheless, there are not any significant differences between 4 and 6 minimal leaf size according to AUC value. The selection gain ratio, information gain, and gini index criterion do not affect the performance results for used dataset. The default value of the “apply pre-running” option is true. If “apply pre-running” option set to false or more than 5 minimal leaf size, the value of AUC and accuracy results decrease.

For Decision Tree, confusion matrix result for 2 minimal leaf size is given in Table 4.8. Confusion matrix result for 5 minimum leaf size is given in Table 4.9. ROC curve of 5 minimum leaf size is given in Figure 4.2.

Table 4.8 Confusion matrix of 2 leaf size of Decision Tree in RapidMiner

Accuracy: 72.94%	True 1	True 0	Class Precision
Prediction 1	1349	581	89.90%
Prediction 0	1609	4554	73.89%
Class Recall	45.81%	88.69%	

Table 4.9 Confusion matrix of 5 leaf size of Decision Tree in RapidMiner

Accuracy: 72.82%	True 1	True 0	Class Precision
Prediction 1	1336	581	89.90%
Prediction 0	1822	4557	73.75%
Class Recall	45.17%	88.74%	

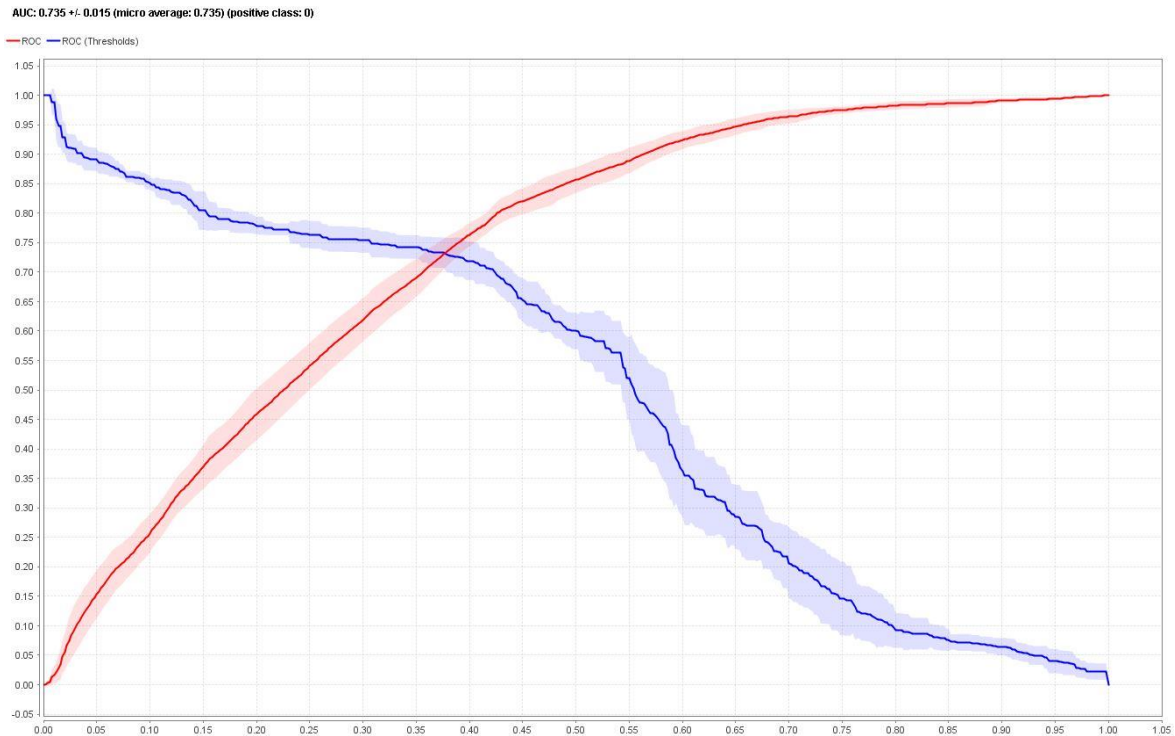


Figure 4.2 ROC curve of 5 leaf size of Decision Tree in RapidMiner

4.2 Random Forest

RF technique is ensemble learning for supervised classification. It uses by constructing multiple decision trees. RF consists of bagging of different decision trees. The feature selection of each split is randomized. Individual predictor strength affects the predictions [31]. RF can handle missing values in the dataset and prevent the overfitting problem. A voting mechanism selects the predictions of whole trees. For example, the majority voting. Screenshot of WEKA tool usage is available in appendix 5 for RF.

The maximum depth of the default value is 0 in Weka. For comparison between RapidMiner and WEKA, RapidMiner maximum depth specified 0 and there were not any

additional filters in both tools in Table 5.2 and Table 5.3. Values of the improved attribute to achieve higher AUC performance is given in Table 4.10 for WEKA.

4.2.1 Random Forest analyses in WEKA

Table 4.10 Default and improved iterations count of RF in WEKA

Settings	Default Value	Improvement
Number of Iterations	100	500

The default value of the number of iterations is 100. The iteration parameter affects the results as positive. Iterations count provide number of trees. According to experiments, the iterations should be 500 to achieve a better AUC result. The differences between the performance results of iterations count are given in Table 4.11.

Table 4.11 Performance results of iterations count of RF in WEKA

Number of Iterations	Accuracy	AUC	Precision	Recall	MAE	RMSE
100 (Default)	76.94%	0.812	0.766	0.769	0.321	0.402
300	76.94%	0.815	0.767	0.769	0.321	0.401
500	77.07%	0.816	0.768	0.771	0.320	0.400
600	76.93%	0.810	0.767	0.769	0.320	0.400

The experiment results show that number of iterations parameter should be 500 to get the highest AUC value for RF in WEKA. Out of the bag calculate the bag errors and verifying the RF model. It is not involved and used in training data. The out of bag score is determined as the correctly predicted rows count from the out of bag example. Therefore, if the dataset is not large enough and uses it totally as the training dataset, the out of bag rate affords can be useful. The default value of “calcOutOfBag” is false. If “calcOutOfBag” options change to true, accuracy and AUC performance results do not change.

Another option is the maximum depth value. Parameter of maximum dept defined the maximum depth of all trees. It described as the longest route between the leaf and the root node. The default value is false, which indicates that each tree will increase until every leaf is pure. It means all the data on the leaf gets from the same class. If this value increases, the

overall performance result decreases for the used dataset. Performance results of maximum depth of tree are given in Table 4.12.

Table 4.12 Performance results of maximum depth of tree count of RF in WEKA

Number of Iterations	Maximum Depth of Tree	Accuracy	AUC	Precision	Recall	MAE	RMSE
100	0	76.94%	0.812	0.766	0.769	0.321	0.402
100	5	72.71%	0.748	0.733	0.727	0.391	0.432

The default value of maximum depth is 0 to provide the best performance results. Nevertheless, there are not any significant differences between 300 and 500 iteration according to AUC value. From these results, it is clear that performance results decrease if more than 500 iterations are available. Comparison of 100 and 500 iterations ROC curves are given in Figure 4.3. Confusion matrix of 100 iteration is given in Table 4.13 and 500 iteration is given in Table 4.14.

Table 4.13 Confusion matrix of 100 iteration of RF in WEKA

Classified As	a	b
a = 0	4517	564
b = 1	1302	1656

Data are classified 77% correctly. According to the confusion matrix, 564 instances should be non-defect but classified as defective and 1302 instances should be defective but classified as non-defective.

Table 4.14 Confusion matrix of 500 iteration of RF in WEKA

Classified As	a	b
a = 0	4590	545
b = 1	1310	1648

Data are classified 77% correctly. According to the confusion matrix, 545 instances should be non-defect but classified as defective and 1310 instances should be defective but classified as non-defective.

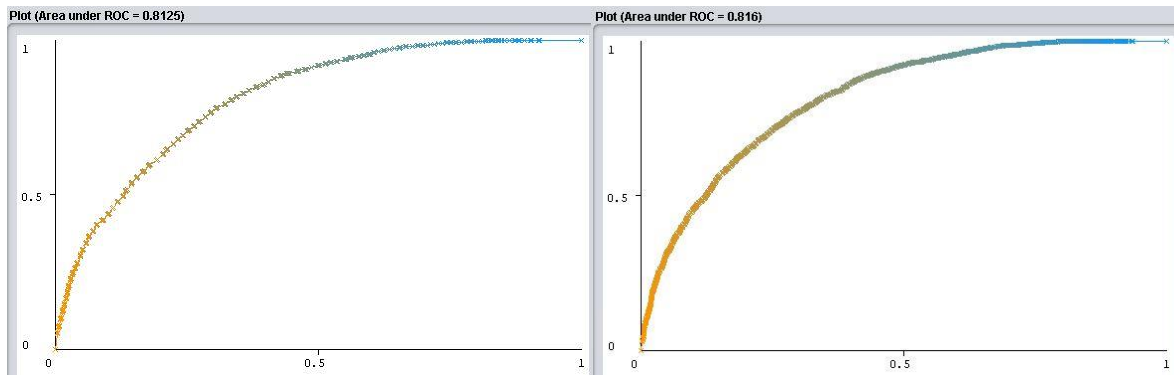


Figure 4.3 ROC curves of 100 and 500 iteration of RF in WEKA

4.2.2 Random Forest analyses in RapidMiner

The criterion of gain ratio, information gain and gini index affect the results. The default criterion in RapidMiner is a gain ratio. Performance results of criterion are given in Table 4.15. Criterion selection means is that a determined split of which attributes will be selected. For default and improvement of RapidMiner configuration, “maximalDept” is set to 0 to compare with the WEKA tool.

Table 4.15 Criterion performance results of RF in RapidMiner

Criterion	Accuracy	AUC	Precision	Recall	MAE	RMSE
Gain Ratio (Default)	74.46%	0.792	0.753	0.889	0.335	0.412
Information Gain	76.62%	0.815	0.772	0.894	0.324	0.401
Gini Index	76.67%	0.813	0.774	0.891	0.325	0.401

Criterion should be information gain to get the highest AUC value for RF. Nevertheless, there are no significant differences between gini index and information gain according to AUC performance result. Another improvement of the RF performance result is the random splits option in RapidMiner. The default option of “random splits” option is disabled. Random splits parameter splits of numerical attributes to be chosen randomly instead of being optimized. If the “random splits” option enabled, the performance result of the AUC value is 0.810. However, still the selection of information gain has the highest performance value for AUC.

For RF, confusion matrix result for gain ratio is given in Table 4.16. Confusion matrix result for information gain is given in Table 4.17. ROC curve of information gain is given in Figure 4.4.

Table 4.16 Confusion matrix of Gain Ratio of RF in RapidMiner

Accuracy: 74.46%	True 1	True 0	Class Precision
Prediction 1	1460	569	71.96%
Prediction 0	1498	4566	75.30%
Class Recall	49.36%	88.92%	

Table 4.17 Confusion matrix of Information Gain of RF in RapidMiner

Accuracy: 76.62%	True 1	True 0	Class Precision
Prediction 1	1608	542	74.79%
Prediction 0	1350	4593	77.28%
Class Recall	54.36%	89.44%	

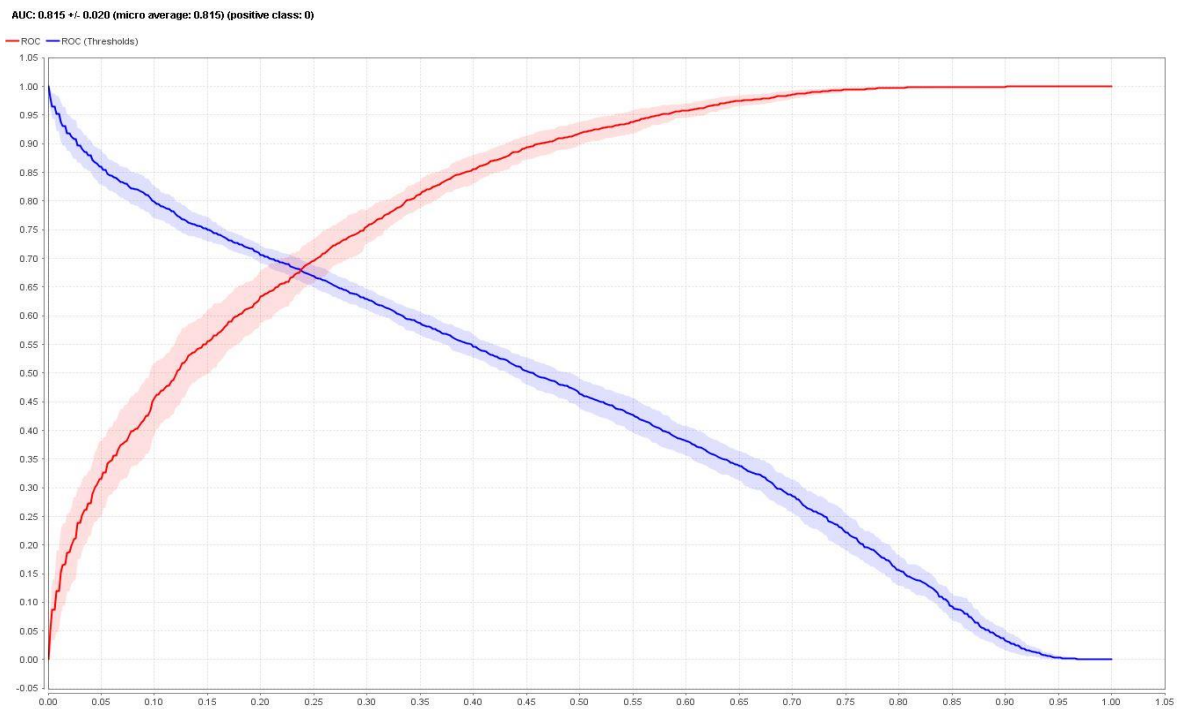


Figure 4.4 ROC curve of Information Gain of RF in RapidMiner

4.3 Bayesian Network

The Bayesian classifier based on Bayes theorem and it is a simple probabilistic classifier. It is independence assumptions between features and predictors. The network form of Bayesian shows graphical attributes description and attributes conditional dependencies. The K2 search algorithm is used in Bayesian Network [26, 28]. Bayesian Network is available in WEKA, but RapidMiner has not Bayesian Network. For this reason, it was analyzed only the WEKA tool. Screenshot of WEKA tool usage is available in appendix 6 for Bayesian Network.

Search algorithms affect to the performance results. The various search algorithms are available in WEKA.

- K2 is a score-based and greedy search algorithm that explores the Bayesian Network structure space to decrease search space. It tries to receive the network structure. K2 makes better learning capability [58].
- Hill climbing iterative algorithm is a heuristic search and mostly applied in Artificial Intelligence for mathematical optimization problems. Hill climbing attempts to find a more suitable solution for optimization difficulties. The most famous example of a hill-climbing problem is the traveling salesman. Traveled measure by salesman needs to decrease.
- LAGDHillClimber is a different Bayes Network learning algorithm. It is the Look Ahead Hill Climbing algorithm. The algorithm does not calculate a best greedy operation such as deleting or adding but estimates a sequence of best greedy operations. Because it is ineffective and slow to consider all the potential arcs.
- RepeatedHillClimber Bayes Network algorithm begins with a randomly created network and then uses hill climber.
- Tabu Search algorithm used for mathematical optimization difficulties and local heuristic search approaches.

K2 greedy search and simple estimator algorithm used with 2 maximum number of parent parameter for default values. Performance results of search algorithms and maximum number of parents are given in Table 4.18.

Table 4.18 Performance results of search algorithms and maximum number of parents

Search Algorithm	Estimator	Result
K2 (Default)	Simple Estimator maxNrOfParents: 2 (Default)	Accuracy: 71.518% AUC: 0.733 Precision: 0.707 Recall: 0.715 MAE: 0.350 RMSE: 0.443
K2	Simple Estimator maxNrOfParents: 3 (Improved)	Accuracy: 72.544% AUC: 0.738 Precision: 0.719 Recall: 0.725 MAE: 0.348 RMSE: 0.437
K2	Simple Estimator maxNrOfParents: 4	Accuracy: 72.309% AUC: 0.735 Precision: 0.716 Recall: 0.723 MAE: 0.345 RMSE: 0.439
Hill Climber	SimpleEstimator maxNrOfParents: 2	Accuracy: 71.802% AUC: 0.732 Precision: 0.710 Recall: 0.718 MAE: 0.353 RMSE: 0.442
LAGDHillClimber	SimpleEstimator maxNrOfParents: 2	Accuracy: 70.666% AUC: 0.707 Precision: 0.698 Recall: 0.707 MAE: 0.385 RMSE: 0.446
RepeatedHillClimber	SimpleEstimator maxNrOfParents: 2	Accuracy: 69.665% AUC: 0.725 Precision: 0.693 Recall: 0.697 MAE: 0.336 RMSE: 0.473
Tabu Search	SimpleEstimator maxNrOfParents: 2	Accuracy: 69.492% AUC: 0.724 Precision: 0.688 Recall: 0.695 MAE: 0.343 RMSE: 0.466

The experiment results display that the K2 search algorithm should be selected with 3 number of parents to get the highest AUC value. Hill Climber, LAGDHillClimber, RepeatedHillClimber and Tabu Search algorithms AUC results lower than K2. Confusion matrix of K2 search algorithm with 2 parent number is given in Table 4.19. Confusion matrix of K2 search algorithm with 3 parent number is given in Table 4.20. Comparison of 2 and 3 parent ROC curves are given in Figure 4.5.

Table 4.19 Confusion matrix of K2 search algorithm with 2 parent number of Bayesian Network

Classified As	a	b
a = 0	4380	755
b = 1	1550	1408

Data are classified 72% correctly. According to the confusion matrix, 755 instances should be non-defect but classified as defective and 1550 instances should be defective but classified as non-defective.

Table 4.20 Confusion matrix of K2 search algorithm with 3 parent number of Bayesian Network

Classified As	a	b
a = 0	4476	659
b = 1	1563	1395

Data are classified 73% correctly. According to confusion matrix, 659 instances should be non-defect but classified as defective and 1563 instance should be defect but classified as non-defective.

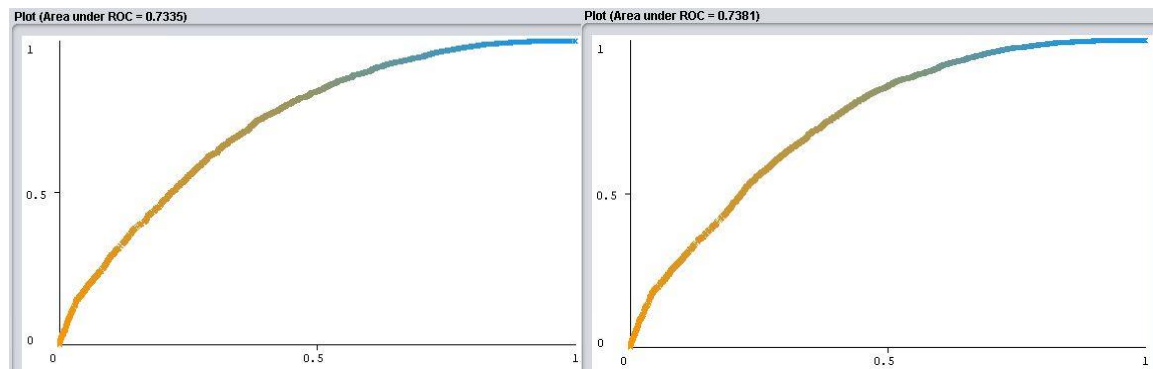


Figure 4.5 ROC curves of K2 search algorithm with 2 and 3 parent number of Bayesian Network

Two different Bayesian network forms are prepared to determine the most effective defect prediction metrics and relationships among all software metrics. The most effective 9 software defect prediction metrics selected after a review of the literature [48].

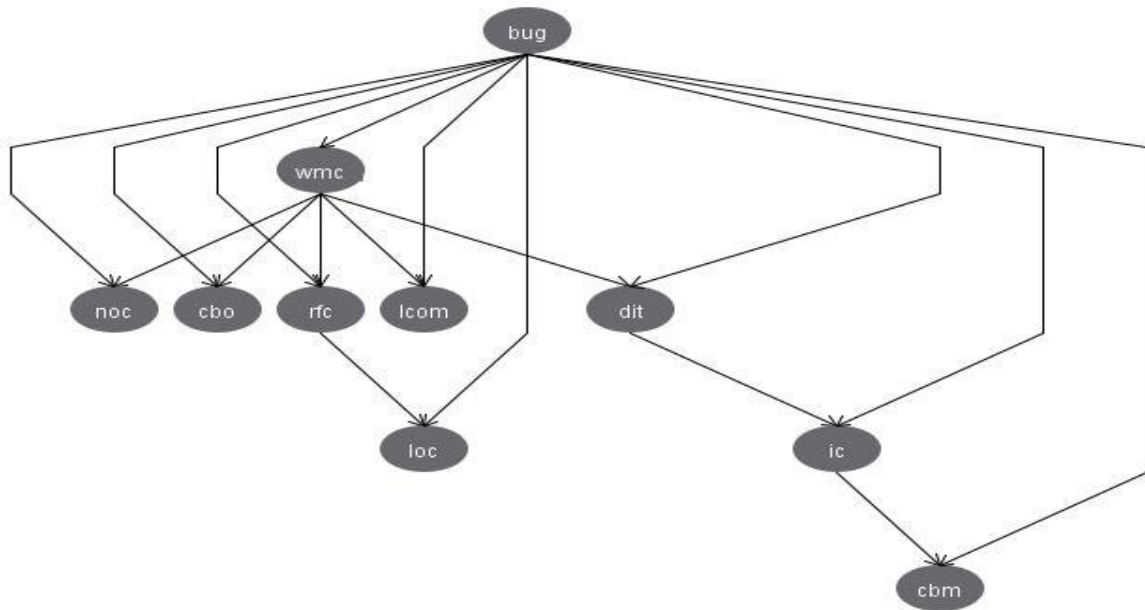


Figure 4.6 Bayesian Network formed for 9 software prediction metrics.

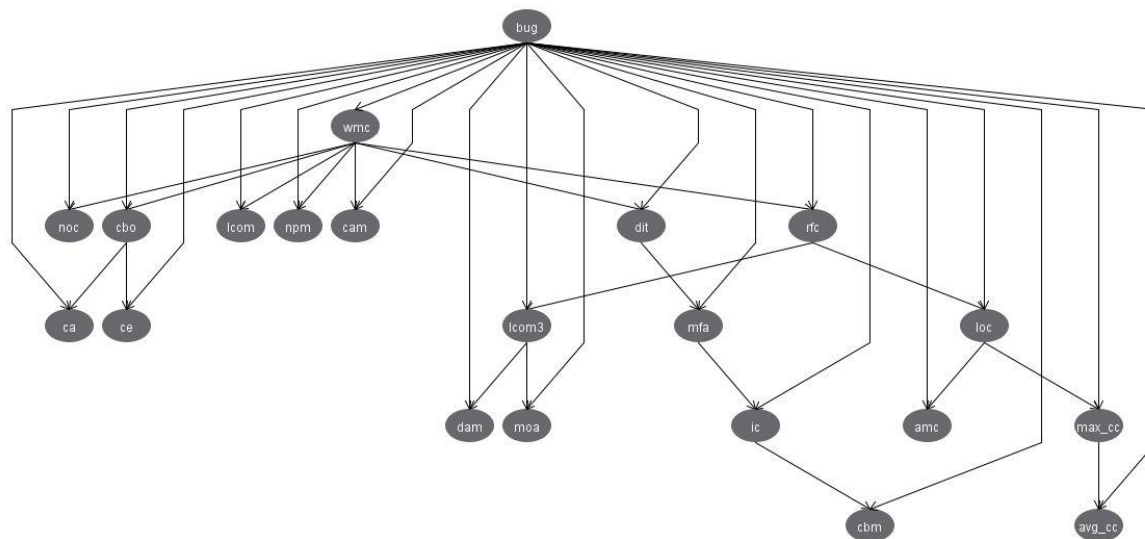


Figure 4.7 Bayesian Network formed for 20 software prediction metrics.

Figure 4.6 and Figure 4.7 display that WMC is the primary useful software metric of defect prediction. Secondary, NOC, CBO, LCOM, NPM, CAM, DIT and RFC software metrics are useful for defect prediction. However, CBM and AVG_CC software metrics are not effective for predicting the defect prone class. Also, DAM, MOA, IC, AMC and MAX_CC software metrics have not significant effectiveness.

4.4 Naïve Bayes

Naïve Bayes is the fastest classification algorithm suitable for a large dataset. Naïve Bayes technique is mostly used in many applications such as text classification, spam filtering and sentiment analysis. It is supervising prediction model that uses Bayes theorem and make classifications of dataset or classify objects based on independent series feature “naive” assumptions. It is widely used and popular for ML techniques. Because the implementation is simple, robustness and effective. The used dataset, software metrics have non-normal distribution. This problem solved with the kernel density estimation of Naïve Bayes [21]. It is effective when the attributes number is high. Screenshot of WEKA tool usage is available in appendix 7 for Naïve Bayes.

Kernel Estimator used in Weka and RapidMiner tools to improve prediction accuracy but default options in WEKA and RapidMiner, Kernel Estimator parameter is false. The kernel is estimation techniques that are used in non-parametric estimation and used in kernel density estimation. It is estimated the random variable probability density function. Values of the improved attribute to achieve higher AUC performance is given in Table 4.21 for WEKA.

4.4.1 Naïve Bayes analyses in WEKA

Using the kernel estimator parameter affects the performance results as positively. The performance results of Kernel Estimator value are given in Table 4.21.

Table 4.21 Kernel Estimator configuration for improving Naïve Bayes performance in WEKA

Settings	Default Value	Improvement
useKernelEstimator	False	True

For Naïve Bayes, performance results of without Kernel Estimator and with Kernel Estimator are given in Table 4.22. Comparison of Kernel Estimator ROC curves are given in Figure 4.8.

Table 4.22 Performance results of Kernel Estimators of Naïve Bayes in WEKA

useKernelEstimator	Accuracy	AUC	Precision	Recall	MAE	RMSE
False (Default)	67.23%	0.676	0.659	0.672	0.329	0.554
True (Improved)	68.36%	0.686	0.670	0.684	0.326	0.517

Confusion matrix of without kernel estimator is given in Table 4.23. Confusion matrix of kernel estimator is given in Table 4.24.

Table 4.23 Confusion matrix of without kernel estimator of Naïve Bayes in WEKA

Classified As	a	b
a = 0	4637	498
b = 1	2154	804

Data are classified 67% correctly. According to the confusion matrix, 498 instances should be non-defect but classified as defective and 2154 instances should be defective but classified as non-defective.

Table 4.24 Confusion matrix of kernel estimator of Naïve Bayes in WEKA

Classified As	a	b
a = 0	4421	714
b = 1	1846	1112

Data are classified 68% correctly. According to the confusion matrix, 714 instances should be non-defect but classified as defective and 1846 instances should be defective but classified as non-defective.

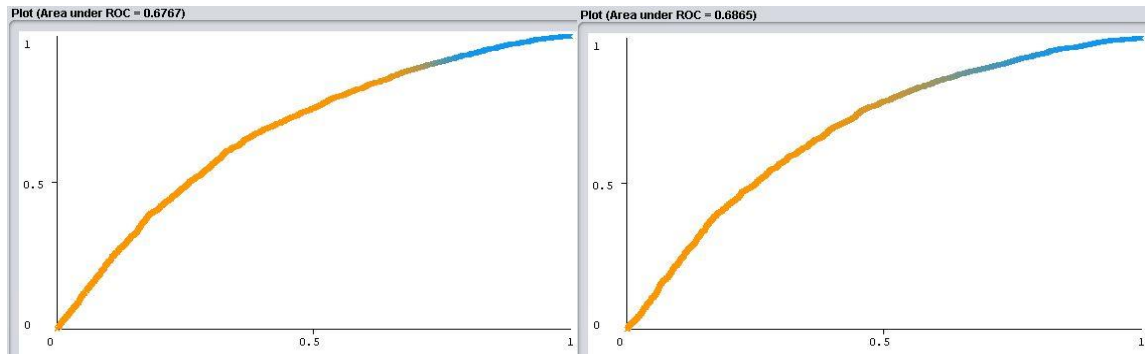


Figure 4.8 ROC curves of kernel estimators of Naïve Bayes in WEKA

4.4.2 Naïve Bayes analyses in RapidMiner

Performance results of Naïve Bayes are given in Table 4.25. Confusion matrix results of Naïve Bayes without kernel are given in Table 4.26. Confusion matrix results of Naïve Bayes Kernel are given in Table 4.27. ROC curve of Naïve Bayes Kernel is given in Figure 4.9. The experiment results show that Naïve Bayes Kernel provides better AUC value.

Table 4.25 Performance results of Naïve Bayes in RapidMiner

Type	Accuracy	AUC	Precision	Recall	MAE	RMSE
Naïve Bayes (Default)	67.07%	0.678	0.682	0.899	0.329	0.555
Naïve Bayes Kernel	67.81%	0.688	0.691	0.890	0.327	0.527

Table 4.26 Confusion matrix of Naïve Bayes in RapidMiner

Accuracy: 67.07%	True 1	True 0	Class Precision
Prediction 1	810	517	61.04%
Prediction 0	2148	4618	68.25%
Class Recall	27.38%	89.93%	

Table 4.27 Confusion matrix of Naïve Bayes Kernel in RapidMiner

Accuracy: 67.81%	True 1	True 0	Class Precision
Prediction 1	916	563	61.93%
Prediction 0	2042	4572	69.13%
Class Recall	30.97%	89.04%	

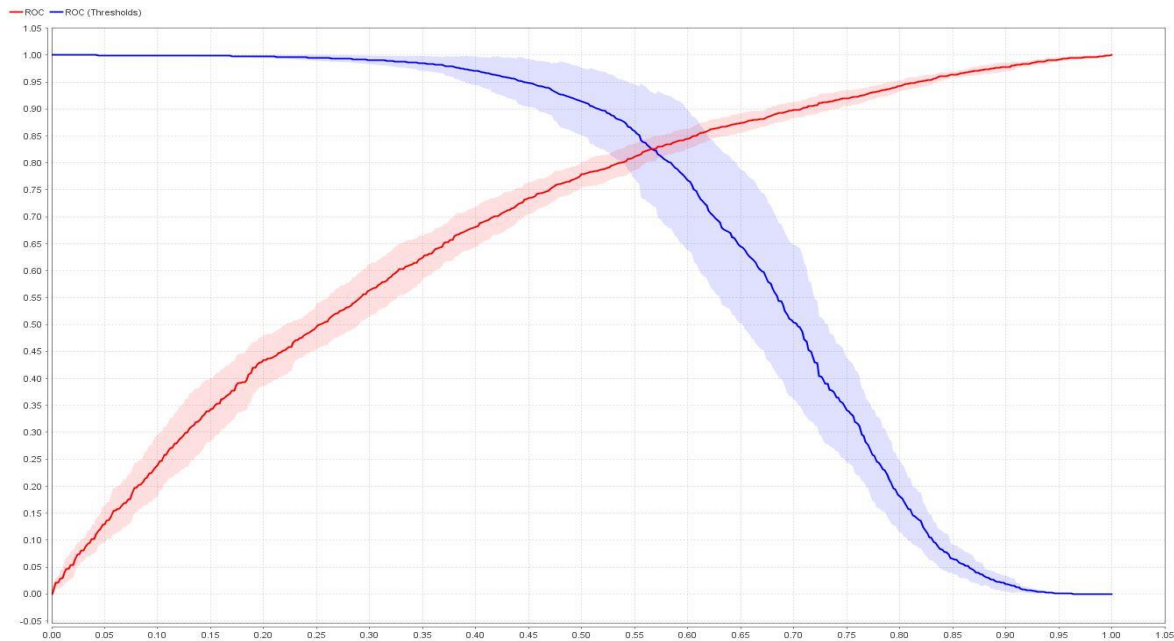


Figure 4.9 ROC curve of Naïve Bayes Kernel in RapidMiner

4.5 Rule based classification

Rule-based classification classifies the records use of IF-THEN rules for class prediction. IF part is condition THEN part is conclusion. Rules are easier to learn than deep trees. One rule is built for each route from the root to a leaf. These classification rules extract from decision trees. Rule-based classification is available as PART in only WEKA [37]. Screenshot of WEKA tool usage is available in appendix 8 for Rule Based classification. Various combinations of the confidence factor, the minimum number of instances per rule and the reduced error-pruning performance result is given in Table 4.28. If the confidence factor set to 0.01 or enable a reduced error pruning option, the prediction accuracy rate decreases.

Table 4.28 Performance results of default and improved configurations of Rule Based Classification

Confidence Factor	Minimum Number of Instance Per Rule	Reduced Error Pruning	Accuracy	AUC	Precision	Recall	MAE	RMSE
0.25 (Default)	2 (Default)	False (Default)	72.55%	0.748	0.718	0.726	0.341	0.434
0.25	3	False	71.61%	0.733	0.720	0.716	0.351	0.441
0.25	4	False	71.29%	0.746	0.704	0.713	0.347	0.438
0.25	2	True	71.11%	0.725	0.702	0.711	0.354	0.446
0.01	2	False	71.95%	0.740	0.717	0.720	0.346	0.439
0.01	2	True	71.11%	0.725	0.702	0.711	0.354	0.446

According to Table 4.28, the default value of WEKA configuration gives the best AUC results. However, there are no crucial differences between 2 and 4 numbers of instances per rule with 0.25 confidence factor. Confusion matrix of 2 minimum number of instances with 0.25 confidence factor is given in Table 4.29. Confusion matrix of 4 minimum number of instances with 0.25 confidence factor is given in Table 4.30. Comparison of 2 and 4 number of instances per rule with 0.25 confidence factor ROC curves are given in Figure 4.10.

Table 4.29 Confusion matrix of 2 minimum number of instances with 0.25 confidence factor of Rule Based classification

Classified As	a	b
a = 0	4397	738
b = 1	1483	1475

Data are classified 73% correctly. According to the confusion matrix, 738 instances should be non-defect but classified as defective and 1483 instances should be defective but classified as non-defective.

Table 4.30 Confusion matrix of 4 minimum number of instances with 0.25 confidence factor of Rule Based classification

Classified As	a	b
a = 0	4445	690
b = 1	1633	1325

Data are classified 71% correctly. According to the confusion matrix, 690 instances should be non-defect but classified as defective and 1633 instances should be defective but classified as non-defective.

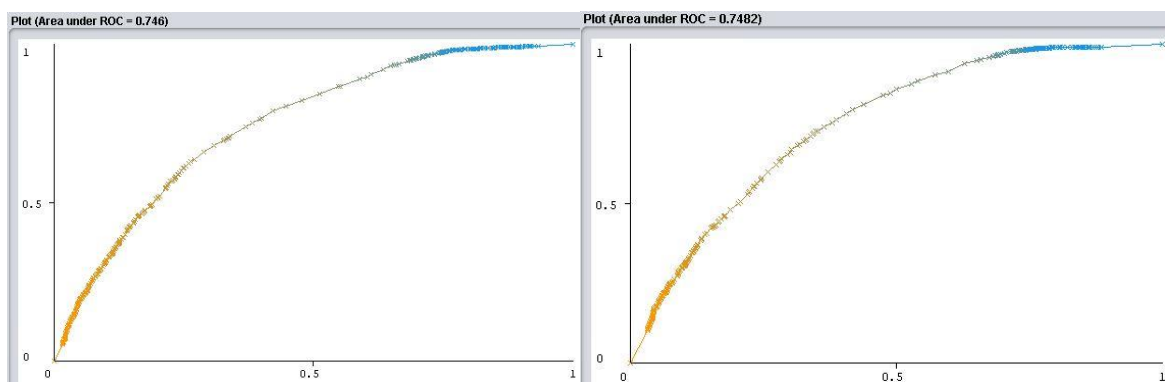


Figure 4.10 ROC curves of 2 and 4 minimum number of instances with 0.25 confidence factor of Rule Based classification

4.6 Support Vector Machine

SVM is used for both classification and regression problems. It is supervised learning model. For finding an optimal decision boundary and non-linear data mapping, the polynomial kernel function is used. SVM tries to find an optimal decision boundary or maximum margin of the plane in the mapped dimension. In short, attempt to find an optimal boundary between the possible outputs. LibLINEAR and a LibSVM are widely used libraries external of WEKA. LibSVM and SMO is non-linear SVMs. Both of them are using a one-vs-one strategy for resolving the multi-class problem. This approach creates a classifier for each couple of classes.

A significant problem is kernel selection in SVM. Thus, analysis focused on various kernel types. The impact of different kernel types analysis provides different performance results. For this reason, kernel types of SMO are essential for improving the performance results.

Kernel function map to data to a higher-dimensional space of feature. Thus, data separation can be more easily and could become a better structure. Feature space of kernel function large and allow flexible conditions. The formulas of kernel functions are given in Table 4.31.

Table 4.31 Kernel functions formulas and parameters

Kernel Functions	Formulas	Parameters
Linear	$K(x, y) = x^T y + c$	c: constant
Polynomial	$K(x, y) = ((x \cdot y) + 1)^p$	p: Polynomial Degree
Normalized Polynomial	$K(x, y) = \frac{((x \cdot y) + 1)^p}{\sqrt{((x \cdot x) + 1)^p ((y \cdot y) + 1)^p}}$	p: Polynomial Degree
Radial Basis Function	$K(x, y) = e^{-\gamma \ x - y\ ^2}$	γ : Kernel Size
Pearson VII universal kernel function (PUK)	$K(x, y) = \frac{1}{\left[1 + \left(\frac{2 \sqrt{\ x - y\ ^2 \sqrt{2(1/\omega) - 1}}}{\sigma} \right)^2 \right]}$	ω, σ : Pearson width
Sigmoid	$K(x, y) = \tanh(ax^T y + c)$	c: constant
Dot	$K(x, y) = x \cdot y$ i.e	-

- Linear is the purest kernel function type. If data is linearly separable, the linear function can be used. It separates data using a single line [11, 54].
- Polynomial is a not fixed kernel. If all training data is normalized, a polynomial kernel is a good choice. It is useful for non-linear model learning. It is often used in Speech Recognition. For a more flexible decision boundary, higher degree kernels of polynomial should be used [20].
- Radial Basis Function, also known as Gaussian Kernel and most used in computer vision [11, 54].
- PUK can be an alternative to linear, polynomial and radial basis kernel functions and solve regression problems [20]. Karl Pearson developed it.
- The sigmoid kernel is also known as Hyperbolic Tangent and Multilayer Perceptron kernel and mostly used in image classification.
- Dot is an inner product. It is non-stationary and can be obtained from linear regression. It is a simple model of the similarity measure.

SVM answers a quadratic programming difficulty. SMO divides the problem into various smaller quadratic problems. It is linear SVM. The binary classifier of the SMO algorithm is available in WEKA [37]. In WEKA, the default kernel type of SMO is polynomial. For comparing with WEKA and RapidMiner results, default kernel type set to polynomial in RapidMiner tool. There were not any additional changes for both tools. Screenshot of WEKA tool usage available in appendix 9 for SMO.

4.6.1 Sequential Minimal Optimization analyses in WEKA

Using the kernel type affects the performance results. For improvement of the SMO performance result, the PUK kernel type should be selected to achieve a better performance result in WEKA. Other performance results of the kernel types are given in Table 4.32. The calibrator of SMO does not affect results.

Table 4.32 Performance results of kernel types of SMO in WEKA

Calibrator	Kernel Type	Results
Gaussian, LibSVM, LinearRegression,	Polynomial (Default)	Accuracy: 68.04% AUC: 0.575 Precision: 0.705 Recall: 0.680 MAE: 0.319 RMSE: 0.565
Logistic, MultilayerPerceptron, SGD,	Normalized Polynomial	Accuracy: 70.04% AUC: 0.606 Precision: 0.721 Recall: 0.700 MAE: 0.299 RMSE: 0.547
SimpleLinearRegression, SimpleLogistic, SMO,	PUK (Improved)	Accuracy: 73.18% AUC: 0.661 Precision: 0.736 Recall: 0.732 MAE: 0.268 RMSE: 0.517
SMOreg, VotedPerceptron	Radial Basis Function	Accuracy: 63.51% AUC: 0.501 Precision: 0.647 Recall: 0.635 MAE: 0.364 RMSE: 0.604

Radial Basis Function kernel type decreases the performance results when compare with Polynomial kernel type. Nevertheless, other kernel types such as Normalized Polynomial, PUK increasing the performance results.

Confusion matrix of Polynomial kernel is given in Table 4.33 and PUK kernel is given in Table 4.34. Comparison of PUK and Polynomial kernel type ROC curves are given in Figure 4.11.

Table 4.33 Confusion matrix of Polynomial kernel of SMO in WEKA

Classified As	a	b
a = 0	4966	169
b = 1	2417	541

Data are classified 68% correctly. According to the confusion matrix, 169 instances should be non-defect but classified as defective and 2417 instances should be defective but classified as non-defective.

Table 4.34 Confusion matrix of PUK kernel of SMO in WEKA

Classified As	a	b
a = 0	4740	395
b = 1	1775	1183

Data are classified 73% correctly. According to confusion matrix, 395 instances should be non-defect but classified as defective and 1775 instance should be defect but classified as non-defective.

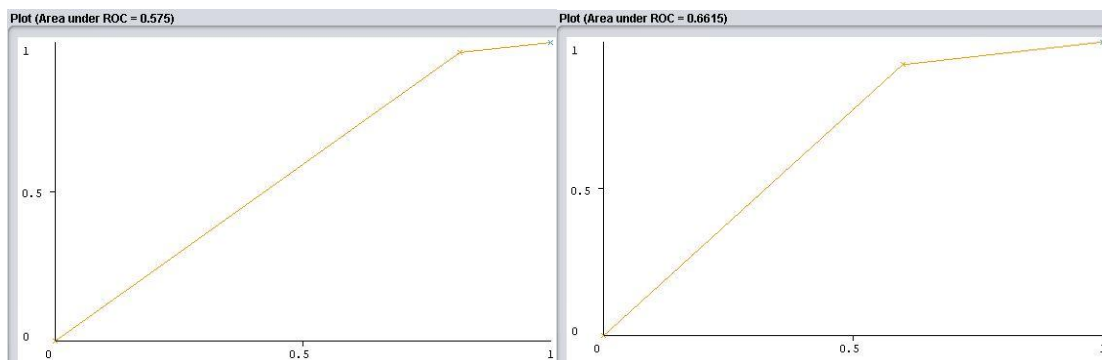


Figure 4.11 ROC curves of Polynomial and PUK kernels of SMO in WEKA

4.6.2 Sequential Minimal Optimization analyses in RapidMiner

For the improvement of the SMO result, the dot kernel type selected in RapidMiner. Other performance results of the kernel types are given in Table 4.35. Dot kernel type increases the performance results when compare with Polynomial kernel type.

Table 4.35 Performance results of kernel types of SMO in RapidMiner

Kernel Type	Results
Polynomial (Default)	Accuracy: 64.88% AUC: 0.666 Precision: 0.669 Recall: 0.900 MAE: 0.423 RMSE: 0.477
Dot (Improved)	Accuracy: 68.82% AUC: 0.691 Precision:0.681 Recall: 0.953 MAE: 0.405 RMSE: 0.456

Confusion matrix of Polynomial kernel is given in Table 4.36 and Dot kernel is given in Table 4.37. ROC curve is given in Figure 4.12 for Dot kernel.

Table 4.36 Confusion matrix of Polynomial kernel of SMO in RapidMiner

Accuracy: 64.88%	True 1	True 0	Class Precision
Prediction 1	627	511	55.10%
Prediction 0	2331	4624	66.48%
Class Recall	21.20%	90.05%	

Table 4.37 Confusion matrix of Dot kernel of SMO in RapidMiner

Accuracy: 68.82%	True 1	True 0	Class Precision
Prediction 1	674	239	73.82%
Prediction 0	2284	4896	68.19%
Class Recall	22.79%	95.35%	

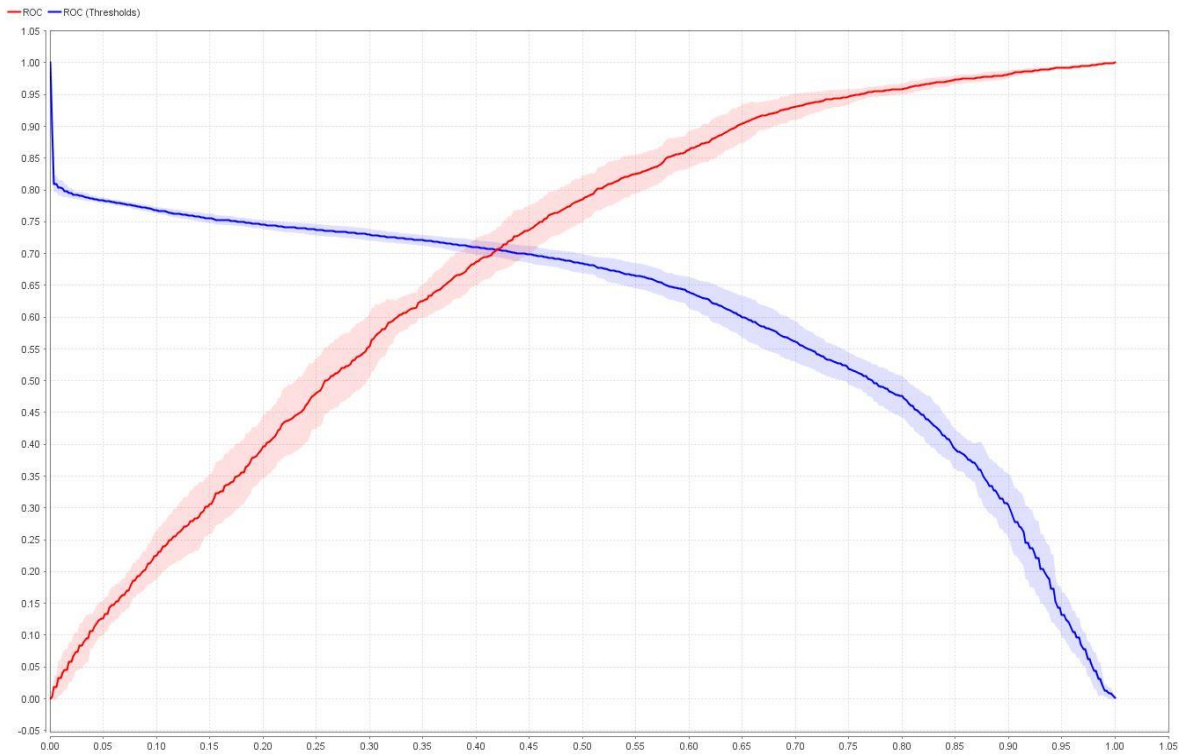


Figure 4.12 ROC curve of Dot kernel of SMO in RapidMiner

4.7 Library for Support Vector Machines

LibSVM is a one-class classification and it is non-linear SVMs. It provides regression and classification. LibSVM implements the SMO method. For both tools, the default kernel type is Radial Basis. Thus, there were not any additional changes in RapidMiner for comparing each other. Screenshot of WEKA tool usage is available in appendix 10 for LibSVM.

4.7.1 Library for Support Vector Machines analyses in WEKA

For the improvement of the LibSVM performance result, the Linear kernel type was selected in WEKA. Performance results of other kernel types analyses are given in Table 4.38.

Table 4.38 Performance results of kernel types of LibSVM in WEKA

Kernel Type	Results
Radial Basis (Default)	Accuracy: 67.66% AUC: 0.563 Precision: 0.728 Recall: 0.677 MAE: 0.323 RMSE: 0.568
Linear	Accuracy: 68.57% AUC: 0.607 Precision: 0.676 Recall: 0.686 MAE: 0.314 RMSE: 0.560
Polynomial	Accuracy: 50.78% AUC: 0.502 Precision: 0.538 Recall: 0.508 MAE: 0.492 RMSE: 0.701
Sigmoid	Accuracy: 53.49% AUC: 0.494 Precision: 0.531 Recall: 0.535 MAE: 0.465 RMSE: 0.682

Radial Basis Function kernel type decreases the performance results when compared with Linear kernel type. Moreover, other kernel types such as Polynomial and Sigmoid decrease the performance results.

Confusion matrix of Radial Basis kernel type is given in Table 4.39 and Linear kernel type is given in Table 4.40. ROC curves of Radial Basis and Linear kernel type are given in Figure 4.13.

Table 4.39 Performance results of Radial Basis kernel of LibSVM in WEKA

Classified As	a	b
a = 0	5052	83
b = 1	2534	424

Data are classified 68% correctly. According to the confusion matrix, 83 instances should be non-defect but classified as defective and 2534 instances should be defective but classified as non-defective.

Table 4.40 Performance results of Linear kernel of LibSVM in WEKA

Classified As	a	b
a = 0	4617	518
b = 1	2025	933

Data are classified 69% correctly. According to the confusion matrix, 518 instances should be non-defect but classified as defective and 2025 instances should be defective but classified as non-defective.

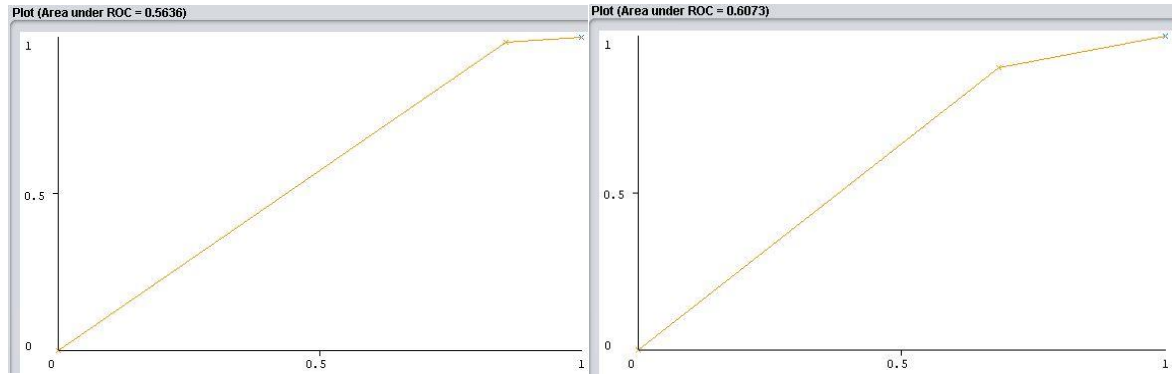


Figure 4.13 ROC curves of Radial Basis and Linear kernels of LibSVM in WEKA

4.7.2 Library for Support Vector Machines analyses in RapidMiner

Radial Basis Kernel is the default value of the RapidMiner tool that provides the best AUC value. However, in WEKA, the Linear kernel type gives the best performance result. Other performance results of kernel types are given in Table 4.41. Confusion matrix of Radial Basis is given in Table 4.42 and ROC curve is given in Figure 4.14.

Table 4.41 Performance results of kernel types of LibSVM in RapidMiner

Kernel Type	Results
Radial Basis (Default)	Accuracy: 66.91% AUC: 0.674 Precision: 0.678 Recall: 0.910 MAE: 0.422 RMSE: 0.463
Linear	Accuracy: 65.34% AUC: 0.659 Precision: 0.650 Recall: 0.982 MAE: 0.426 RMSE: 0.472
Polynomial	Accuracy: 63.50% AUC: 0.645 Precision: 0.634 Recall: 0.999 MAE: 0.437 RMSE: 0.490
Sigmoid	Accuracy: 58.88% AUC: 0.403 Precision: 0.676 Recall: 0.676 MAE: 0.411 RMSE: 0.638

Sigmoid kernel type gives the worst performance results when comparing the other kernel types. There are no significant differences between Radial Basis and Linear kernel type according to AUC value.

Table 4.42 Confusion matrix of Radial Basis kernel of LibSVM in RapidMiner

Accuracy: 66.91%	True 1	True 0	Class Precision
Prediction 1	740	460	61.67%
Prediction 0	2218	4675	67.82%
Class Recall	25.02%	91.04%	

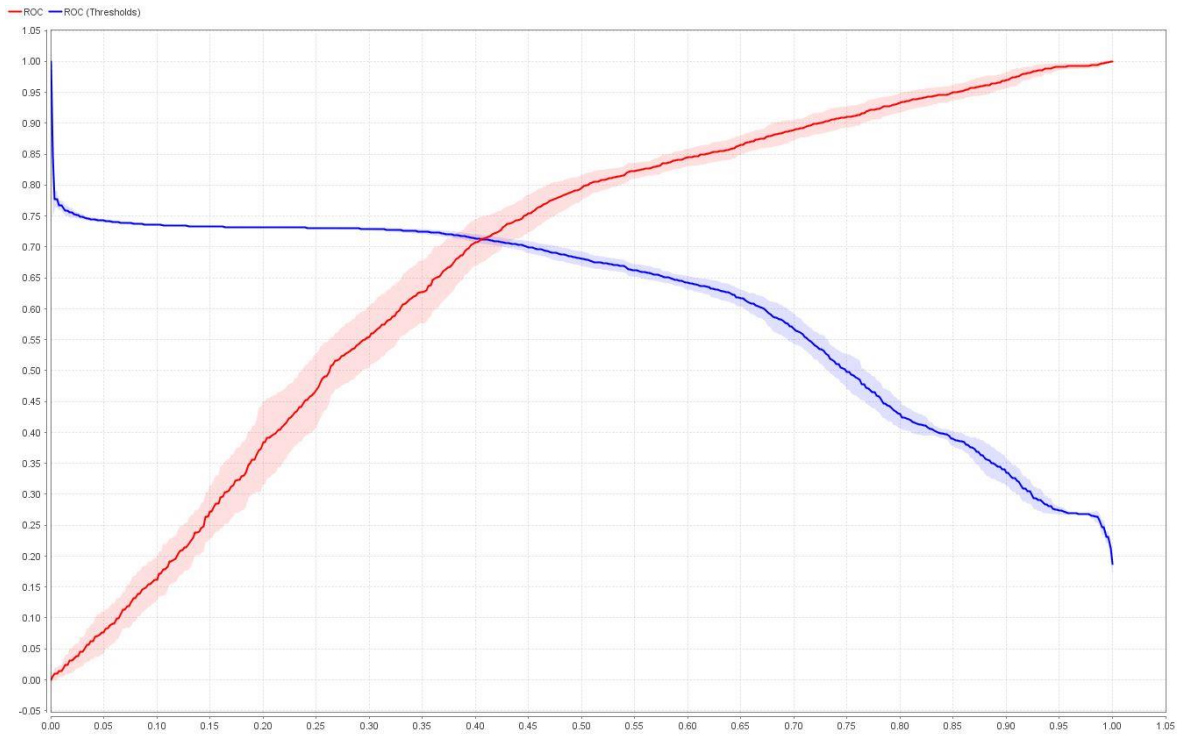


Figure 4.14 ROC curve of Radial Basis kernel of LibSVM in RapidMiner

4.8 Library for large linear classification

LibLINEAR is logistic regression methods and uses L1 norm. It implements linear SVM that answers the classification problem linearly without kernels. This can decrease the training time. Previous researches show that LibLINEAR performance faster than LibSVM and SMO [23]. Prediction accuracy and AUC results effected by SVM type. Due to this, all SVM types analyzed and results are given in Table 4.43. Screenshots of WEKA tool usage available in appendix 11 and for LibLINEAR.

L2-norm is famous for classification problems and most used 2 class classification. L1-norm has higher dimensional feature spaces than L2-norm. It ignores unnecessary

features and puts more weight on the most significant features. L1 regularized logistic regression can solve convex optimization problems.

4.8.1 Library for large linear classification analyses in WEKA

The default SVM type of WEKA for the LibLINEAR algorithm is L2-regularized L2-loss support vector classification (dual) and performance results of analyses are given in Table 4.43. All other SVM types increase the performance results except L2-regularized L2-loss support vector classification (dual) and Support vector classification by Crammer and Singer Multi-class classification when comparing the L2-regularized L2-loss support vector classification (dual) default SVM type. Nevertheless, there are no critical differences between L2-regularized L2-loss support vector classification (dual) and L2-regularized L1-loss support vector classification (dual) according to AUC values. Support vector classification by Crammer and Singer Multi-class classification SVM type has the worst performance results.

Table 4.43 Performance results of SVM types of LibLINEAR in WEKA

SVM Type	Results
L2-regularized logistic regression (primal) Multi-class classification	Accuracy: 69.86 % AUC: 0.619 Precision: 0.695 Recall: 0.699 MAE: 0.301 RMSE: 0.549
L2-regularized L2-loss support vector classification (dual) (Default) Multi-class classification	Accuracy: 64.09% AUC: 0.563 Precision: 0.614 Recall: 0.614 MAE: 0.359 RMSE: 0.599
L2-regularized L2-loss support vector classification (primal) Multi-class classification	Accuracy: 69.59% AUC: 0.611 Precision: 0.696 Recall: 0.696 MAE: 0.304 RMSE: 0.551
L2-regularized L1-loss support vector classification (dual)	Accuracy: 63.30% AUC: 0.562 Precision: 0.608 Recall: 0.633 MAE: 0.367 RMSE: 0.605
Support vector classification by Crammer and Singer Multi-class classification	Accuracy: 63.79% AUC: 0.543 Precision: 0.605 Recall: 0.638 MAE: 0.362 RMSE: 0.601
L1-regularized L2-loss support vector classification Multi-class classification	Accuracy: 69.56% AUC: 0.612 Precision: 0.694 Recall: 0.696 MAE: 0.304 RMSE: 0.551
L1-regularized logistic regression	Accuracy: 69.87% AUC: 0.620 Precision: 0.695 Recall: 0.699 MAE: 0.301 RMSE: 0.548
L2-regularized logistic regression (dual)	Accuracy: 64.89% AUC: 0.614 Precision: 0.644 Recall: 0.649 MAE: 0.351 RMSE: 0.592

The L1-regularized logistic regression model improves prediction accuracy results. Confusion matrix of L2-regularized L2-loss support vector classification (dual) is given in Table 4.44 and L1-regularized logistic regression is given in Table 4.45. Comparison of L2-regularized L2-loss support vector classification (dual) and L1-regularized logistic regression ROC curves are given in Figure 4.15.

Table 4.44 Confusion matrix of L2-Regularized L2-Loss Support Vector Classification (Dual) of LibLINEAR in WEKA

Classified As	a	b
a = 0	4377	758
b = 1	2148	810

Data are classified 64% correctly. According to the confusion matrix, 758 instances should be non-defect but classified as defective and 2148 instances should be defective but classified as non-defective.

Table 4.45 Confusion matrix of L1-Regularized Logistic Regression of LibLINEAR in WEKA

Classified As	a	b
a = 0	4684	451
b = 1	1987	971

Data are classified 77% correctly. According to the confusion matrix, 451 instances should be non-defect but classified as defective and 1987 instances should be defective but classified as non-defective.

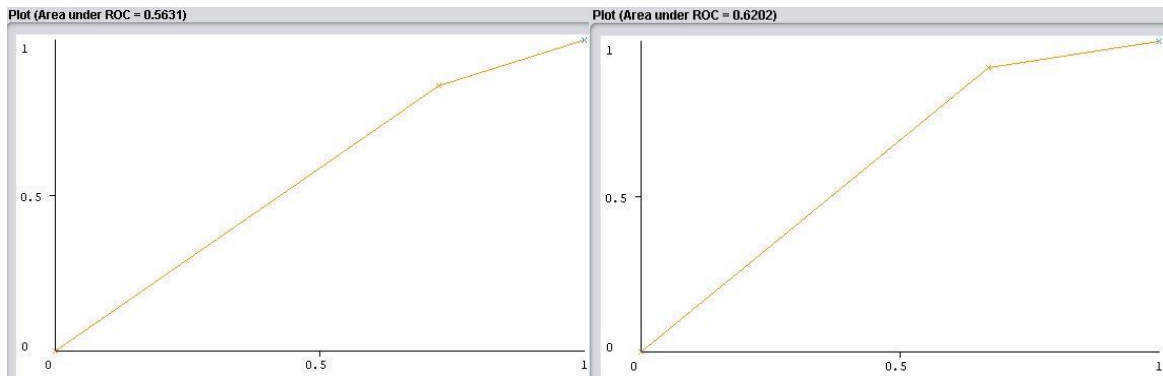


Figure 4.15 ROC curves of L2-Regularized L2-Loss Support Vector Classification (Dual) and L1-Regularized Logistic Regression of LibLINEAR in WEKA

4.8.2 Library for large linear classification analyses in RapidMiner

The default setting of RapidMiner provides the best prediction results. Confusion matrix result is given in Table 4.46. ROC curve is given in Figure 4.16.

Table 4.46 Confusion matrix of LibLINEAR default configurations in RapidMiner

Accuracy: 68.97%	True 1	True 0	Class Precision
Prediction 1	582	235	74.37%
Prediction 0	2276	4900	68.28%
Class Recall	23.06%	95.42%	

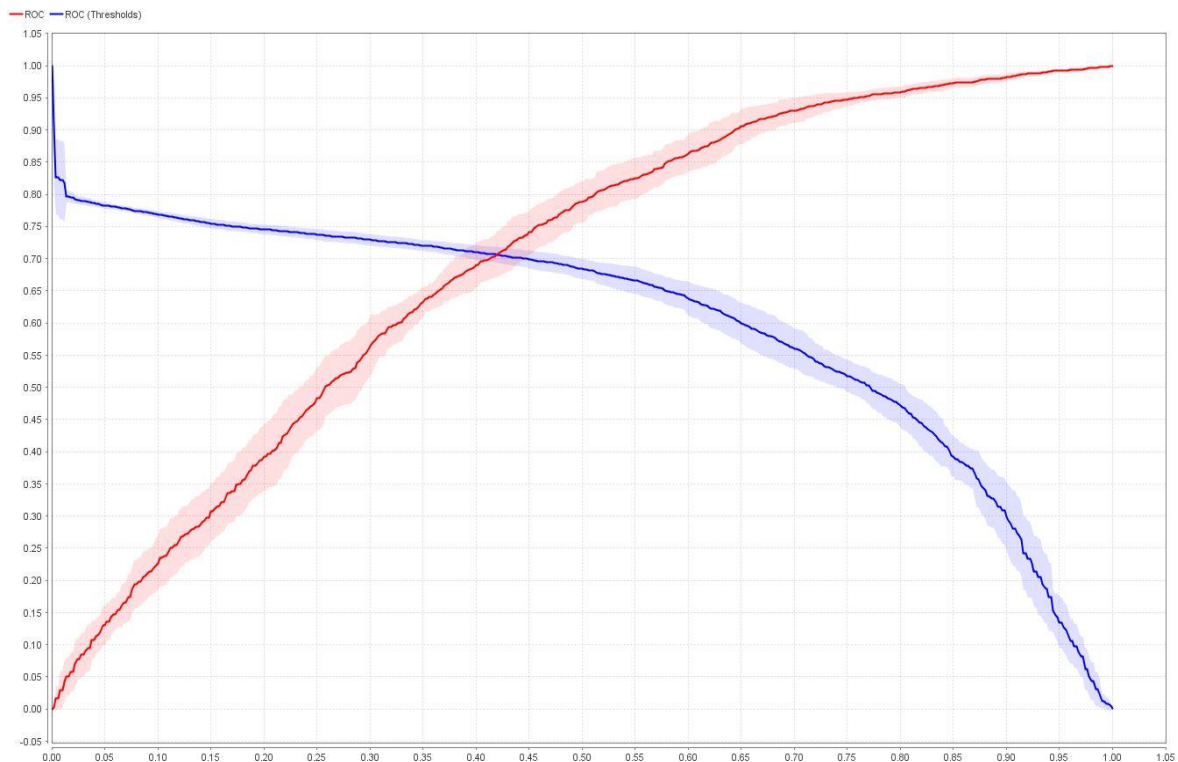


Figure 4.16 ROC curve of LibLINEAR default configurations in RapidMiner

4.9 Logistic Regression

LR algorithm is a predictive regression analysis based on a statistical model and it is supervised learning classification. It is beneficial for binary classification problems and categorical target value. It is a binary regression form. Screenshots of WEKA tool usage available in appendix 12 for LR.

The default value of both tools provides the best results. For comparison between RapidMiner and WEKA, RapidMiner maximum iteration specified -1 because the default value of maximum iteration is -1 in WEKA. There were not any additional filters in both tools in Table 5.2 and Table 5.3. Changing of maximum iterations decreases performance results.

4.9.1 Logistic Regression analyses in WEKA

The default value of the number of decimal places is 4 in WEKA. Changing of decimal places count do not affect to performance results. Confusion matrix of default configurations is given in Table 4.47 and ROC curve is given in Figure 4.17 for WEKA.

Table 4.47 Confusion matrix of default configurations of LR in WEKA

Classified As	a	b
a = 0	4685	450
b = 1	1982	976

Data are classified 70% correctly. According to the confusion matrix, 450 instances should be non-defect but classified as defective and 1982 instances should be defective but classified as non-defective.

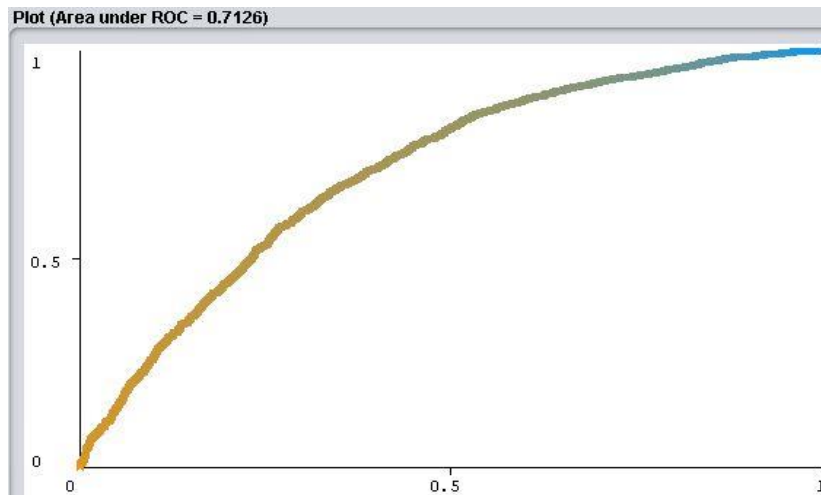


Figure 4.17 ROC curve of default configurations of LR in WEKA

4.9.2 Logistic Regression analyses in RapidMiner

The default values of the LR model provide the best AUC value. For LR, confusion matrix result of RapidMiner is given in Table 4.48 and ROC curve is given in Figure 4.18.

Table 4.48 Confusion matrix of default configuration of LR in RapidMiner

Accuracy: 69.86%	True 1	True 0	Class Precision
Prediction 1	980	461	68.01%
Prediction 0	1978	4674	70.26%
Class Recall	33.13%	91.02%	

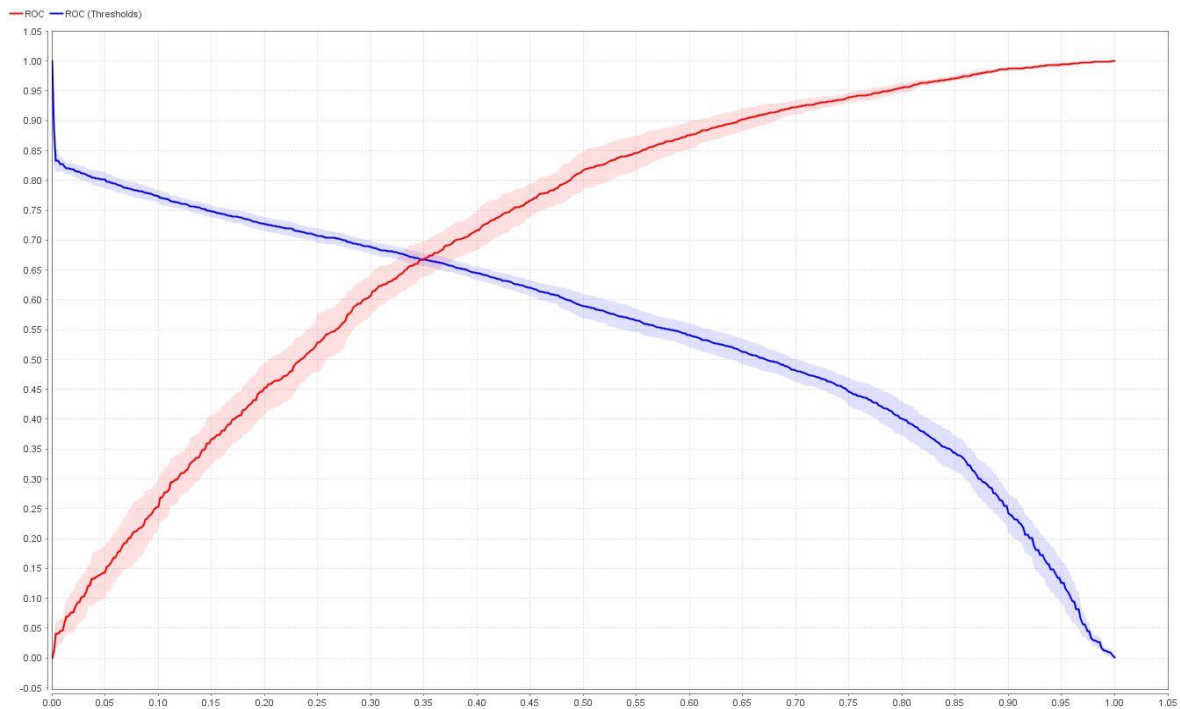


Figure 4.18 ROC curve of default configuration of LR in RapidMiner

4.10 Bagging

Bagging is a bootstrap aggregation. It is an ensemble model that creates a strong learner with a combined group of weak learners. An ensemble approach combines the multiple ML techniques together. Thus, prediction accuracy performance better than the individual ML techniques. As a result, weak learner variance and bias reduce for better performance. For that reason, this model improves stability and increases robustness. Also, it helps to avoid overfitting [33]. Differences from Boosting; Bagging algorithm tries to overcome over-fitting difficulty. If the classifier is unstable or high variance, bagging should be selected. The bagging weight is equal for each model. It is trying to decrease the model's variance.

As a result, the accuracy of the bagging result is high. The default classifier of the Bagging algorithm is Regression Tree, also called REPTree in WEKA. For comparison between RapidMiner and WEKA tools, the Decision Tree classifier was selected in the Boosting algorithm in RapidMiner. Nevertheless, if the RF classifier is selected instead of Regression Tree, the best AUC result is obtained for both tools. There were no additional filters in both tools in Table 5.2 and Table 5.3. Screenshots of WEKA tool usage in appendix 13 for Bagging.

4.10.1 Bagging analyses in WEKA

Random Forest classifier in bagging affects the results as positive. However, performance results of other classifiers are given in Table 4.49 for WEKA. The default settings of WEKA for the bagging algorithm is REPTree and confusion matrix is given in Table 4.50. The best classifier is the RF. Confusion matrix of RF is given in Table 4.51.

Table 4.49 Performance results of Bagging classifiers in WEKA

Classifier	Results
REPTree (Default)	Accuracy: 75.22% AUC: 0.786 Precision: 0.749 Recall: 0.752 MAE: 0.335 RMSE: 0.413
Random Tree	Accuracy: 74.96 % AUC: 0.786 Precision: 0.746 Recall: 0.750 MAE: 0.320 RMSE: 0.417
Random Forest (Improved)	Accuracy: 76.59% AUC: 0.811 Precision: 0.764 Recall: 0.766 MAE: 0.329 RMSE: 0.402
LMT	Accuracy: 74.64 % AUC: 0.791 Precision: 0.741 Recall: 0.746 MAE: 0.316 RMSE: 0.415
J48	Accuracy: 74.83% AUC: 0.789 Precision: 0.743 Recall: 0.748 MAE: 0.319 RMSE: 0.414
Decision Stump	Accuracy: 66.52% AUC: 0.653 Precision: 0.669 Recall: 0.665 MAE: 0.438 RMSE: 0.464

In the experiments, REPTree and Random Tree classifiers have the same AUC value. Although REPTree accuracy value better than Random Tree.

Decision stump is a less effective classifier compared to others. ROC curves of REPTree and RF are given in Figure 4.19.

Table 4.50 Confusion matrix of REPTree in Bagging for WEKA

Classified As	a	b
a = 0	4569	566
b = 1	1439	1519

Data are classified 75% correctly. According to the confusion matrix, 566 instances should be non-defect but classified as defective and 1439 instances should be defective but classified as non-defective.

Table 4.51 Confusion matrix of RF in Bagging for WEKA

Classified As	a	b
a = 0	4611	524
b = 1	1370	1588

Data are classified 77% correctly. According to the confusion matrix, 524 instances should be non-defect but classified as defective and 1370 instances should be defective but classified as non-defective.

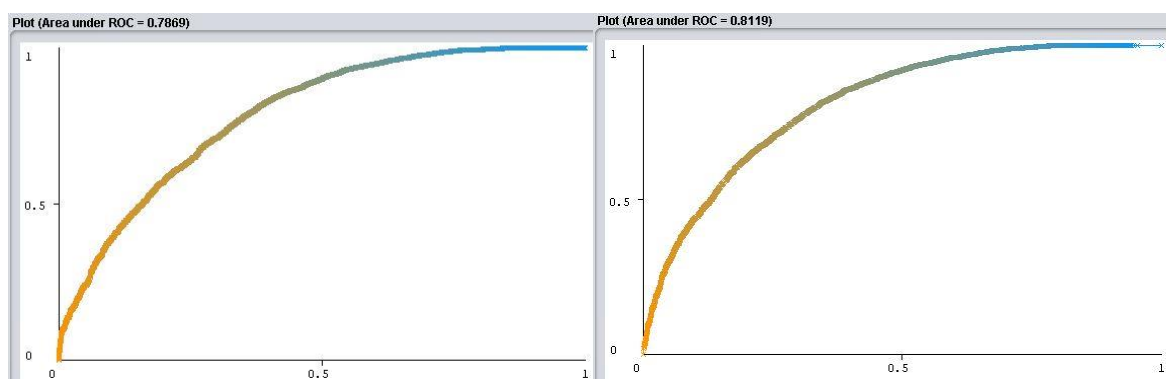


Figure 4.19 ROC curves of REPTree and Random Forest in Bagging for WEKA

4.10.2 Bagging analyses in RapidMiner

RF classifier in bagging affects the results as positive too. The criterion of information gain and gini index provides the same AUC results for RF. Information gain and gini index criterion does not affect Random Tree performance results. However, other classifiers and criterion analysis results are given in Table 4.52 for RapidMiner.

Table 4.52 Performance results of Bagging classifiers in RapidMiner

Classifier	Criterion	Results
Decision Tree	Gain Ratio	Accuracy: 65.95% AUC: 0.590 Precision: 0.655 Recall: 0.981 MAE: 0.444 RMSE: 0.469
Decision Tree (Default)	Information Gain (Default)	Accuracy: 73.14% AUC: 0.746 Precision: 0.726 Recall: 0.926 MAE: 0.363 RMSE: 0.429
Decision Tree	Gini Index	Accuracy: 73.58% AUC: 0.764 Precision: 0.737 Recall: 0.907 MAE: 0.353 RMSE: 0.424
Random Tree	Gain Ratio, Information Gain	Accuracy: 63.45% AUC: 0.500 Precision: 0.634 MAE: 0.464 RMSE: 0.482
Random Tree	Gini Index	Accuracy: 70.04% AUC: 0.717 Precision: 0.698 Recall: 0.930 MAE: 0.419 RMSE: 0.449
Random Forest	Gain Ratio	Accuracy: 65.28% AUC: 0.698 Precision: 0.647 Recall: 0.992 MAE: 0.445 RMSE: 0.467
Random Forest (Improved)	Information Gain (Improved)	Accuracy: 74.66% AUC: 0.790 Precision: 0.738 Recall: 0.930 MAE: 0.361 RMSE: 0.415
Random Forest	Gini Index	Accuracy: 74.68% AUC: 0.790 Precision: 0.739 Recall: 0.929 MAE: 0.361 RMSE: 0.414
Decision Stump	Gain Ratio	Accuracy: 64.34% AUC: 0.515 Precision: 0.640 Recall: 0.998 MAE: 0.458 RMSE: 0.478
Decision Stump	Information Gain	Accuracy: 67.66% AUC: 0.613 Precision: 0.675 Recall: 0.943 MAE: 0.438 RMSE: 0.467
Decision Stump	Gini Index	Accuracy: 67.66% AUC: 0.606 Precision: 0.675 Recall: 0.943 MAE: 0.438 RMSE: 0.467

Confusion matrix of Decision Tree classifier is given in Table 53. Confusion matrix of RF classifier is given in Table 4.54 and ROC curve is given in Figure 4.20.

Table 4.53 Confusion matrix of Decision Tree classifier in Bagging for RapidMiner

Accuracy: 73.14%	True 1	True 0	Class Precision
Prediction 1	1162	378	75.45%
Prediction 0	1796	4757	72.59%
Class Recall	39.28%	92.64%	

Table 4.54 Confusion matrix of RF classifier in Bagging for RapidMiner

Accuracy: 74.66%	True 1	True 0	Class Precision
Prediction 1	1264	357	77.98%
Prediction 0	1694	4778	73.83%
Class Recall	42.73%	93.05%	

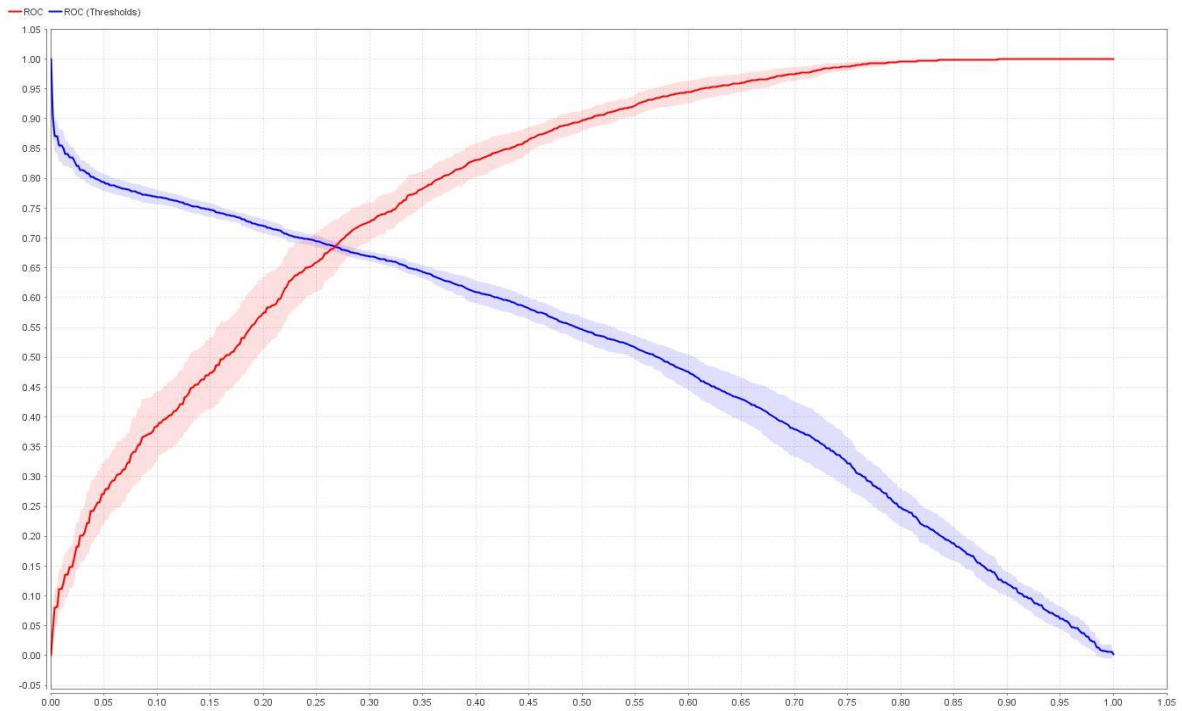


Figure 4.20 ROC curve of RF in Bagging for RapidMiner

4.11 Boosting

Boosting is another ensemble algorithm. It changes weak learners to strong learners. They have similar definitions with Bagging. Differences from Bagging; Boosting algorithm tries to overcome bias. If the classifier is simple or high bias, boosting should be selected. The weight of boosting calculates according to their performance results. It is trying to decrease the model's bias. It is implemented as AdaBoost in Weka. Adaboost joins various weak learners into a single strong learner. In WEKA, the default classifier of Boosting algorithms is Decision Stump. For comparison between RapidMiner and WEKA tools, the Decision Stump classifier was selected in the Boosting algorithm in RapidMiner. In the experiments, the RF classifier should be selected instead of Decision Stump for best AUC results in both tools and there were no additional filters in Tables 5.2 and 5.3. As is seen in Table 4.55 and Table 4.56, RF algorithms provide the best results for prediction accuracy for Weka and RapidMiner. Screenshots of WEKA tool usage in appendix 14 for Boosting.

4.11.1 Boosting analyses in WEKA

RF classifier affects the results as positive in boosting algorithm. However, performance results of other classifiers results are given in Table 4.55 for WEKA. The default settings of WEKA for the boosting algorithm is Decision Stump and the confusion matrix is given in Table 4.56. Confusion matrix of RF are given in Table 4.57. The best classifier is the RF same as the bagging algorithm.

Table 4.55 Performance results of Boosting classifiers in WEKA

Classifier	Results
Decision Stump (Default)	Accuracy: 70.29% AUC: 0.712 Precision: 0.698 Recall: 0.703 MAE: 0.395 RMSE: 0.445
Random Tree	Accuracy: 73.12% AUC: 0.766 Precision: 0.725 Recall: 0.731 MAE: 0.276 RMSE: 0.492
Random Forest	Accuracy: 76.62% AUC: 0.781 Precision: 0.763 Recall: 0.766 MAE: 0.238 RMSE: 0.471
LMT	Accuracy: 72.38% AUC: 0.765 Precision: 0.719 Recall: 0.724 MAE: 0.281 RMSE: 0.489
J48	Accuracy: 72.82% AUC: 0.768 Precision: 0.724 Recall: 0.728 MAE: 0.279 RMSE: 0.483
REPTree	Accuracy: 73.34% AUC: 0.760 Precision: 0.728 Recall: 0.733 MAE: 0.297 RMSE: 0.462

In the experiments, REPTree and Random Tree classifiers have the close AUC value the same as the bagging algorithm. ROC curves of Decision Stump and RF are given in Figure 4.21. The decision stump algorithm is the less effective classifier compared to others.

Table 4.56 Confusion matrix of Decision Stump in Boosting for WEKA

Classified As	a	b
a = 0	4644	491
b = 1	1913	1045

Data are classified 70% correctly. According to the confusion matrix, 491 instances should be non-defect but classified as defective and 1913 instances should be defective but classified as non-defective.

Table 4.57 Confusion matrix of RF in Boosting for WEKA

Classified As	a	b
a = 0	4579	556
b = 1	1336	1622

Data are classified 77% correctly. According to the confusion matrix, 556 instances should be non-defect but classified as defective and 1336 instances should be defective but classified as non-defective.

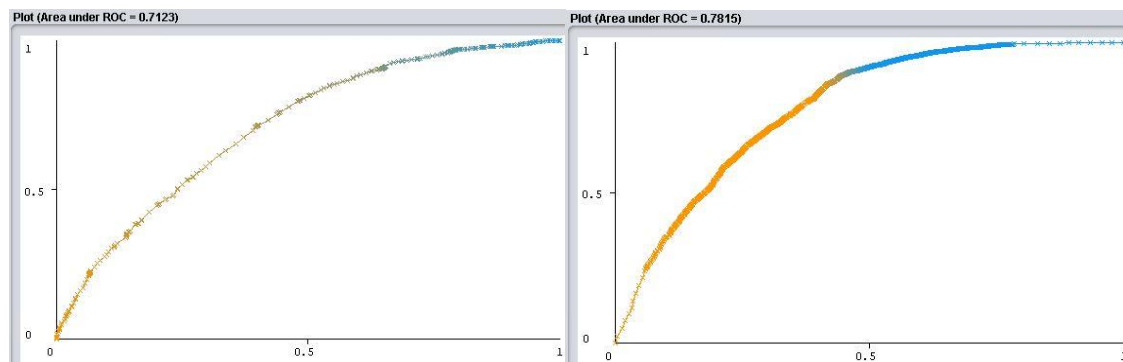


Figure 4.21 ROC curves of Decision Stump and RF in Boosting for WEKA

4.11.2 Boosting analyses in RapidMiner

RF classifier and information gain criterion affect the results as positive too. The criterion of information gain and gini index provides the same AUC results for Random Tree. Information gain and gini index criterion does not affect Random Tree performance results. However, performance results of other classifiers and criterion are given in Table 4.58 for RapidMiner.

Table 4.58 Performance results of Boosting classifiers in RapidMiner

Classifier	Criterion	Results
Decision Tree	Gain Ratio	Accuracy: 65.57% AUC: 0.552 Precision: 0. 651 Recall: 0. 988 MAE: 0.443 RMSE: 0.473
Decision Tree	Information Gain	Accuracy: 71.83% AUC: 0.656 Precision: 0.718 Recall: 0.915 MAE: 0.380 RMSE: 0.452
Decision Tree	Gini Index	Accuracy: 73.03% AUC: 0.676 Precision: 0. 743 Recall: 0. 879 MAE: 0.366 RMSE: 0.448
Random Tree	Gain Ratio, Information Gain	Accuracy: 63.45% AUC: 0.500 Precision: 0. 634 MAE: 0.464 RMSE: 0.482
Random Tree	Gini Index	Accuracy: 68.73% AUC: 0.667 Precision: 0.698 Recall: 0. 892 MAE: 0.370 RMSE: 0.475
Random Forest	Gain Ratio	Accuracy: 66.12% AUC: 0.547 Precision: 0.654 Recall: 0.990 MAE: 0.443 RMSE: 0.473
Random Forest (Improved)	Information Gain	Accuracy: 74.46% AUC: 0.693 Precision: 0.738 Recall: 0.925 MAE: 0.339 RMSE: 0.443
Random Forest	Gini Index	Accuracy: 74.58% AUC: 0.692 Precision: 0.739 Recall: 0. 926 MAE: 0.334 RMSE: 0.444
Decision Stump	Gain Ratio	Accuracy: 64.34% AUC: 0.513 Precision: 0.640 Recall: 0.999 MAE: 0.459 RMSE: 0.479
Decision Stump	Information Gain	Accuracy: 67.19% AUC: 0.653 Precision: 0. 674 Recall: 0. 933 MAE: 0.414 RMSE: 0.462
Decision Stump (Default)	Gini Index (Default)	Accuracy: 67.66% AUC: 0.655 Precision: 0. 675 Recall: 0. 943 MAE: 0.413 RMSE: 0.461

Confusion matrix of the Decision Stump classifier result is given in Table 4.59 and the RF classifier is given in Table 4.60. ROC curve is given in Figure 4.22.

Table 4.59 Confusion matrix of Decision Stump classifier in Boosting for RapidMiner

Accuracy: 67.76%	True 1	True 0	Class Precision
Prediction 1	630	289	68.55%
Prediction 0	2328	4846	67.55%
Class Recall	21.30%	94.37%	

Table 4.60 Confusion matrix of RF classifier in Boosting for RapidMiner

Accuracy: 74.46%	True 1	True 0	Class Precision
Prediction 1	1273	382	76.92%
Prediction 0	1685	4753	73.83%
Class Recall	43.04%	92.56%	

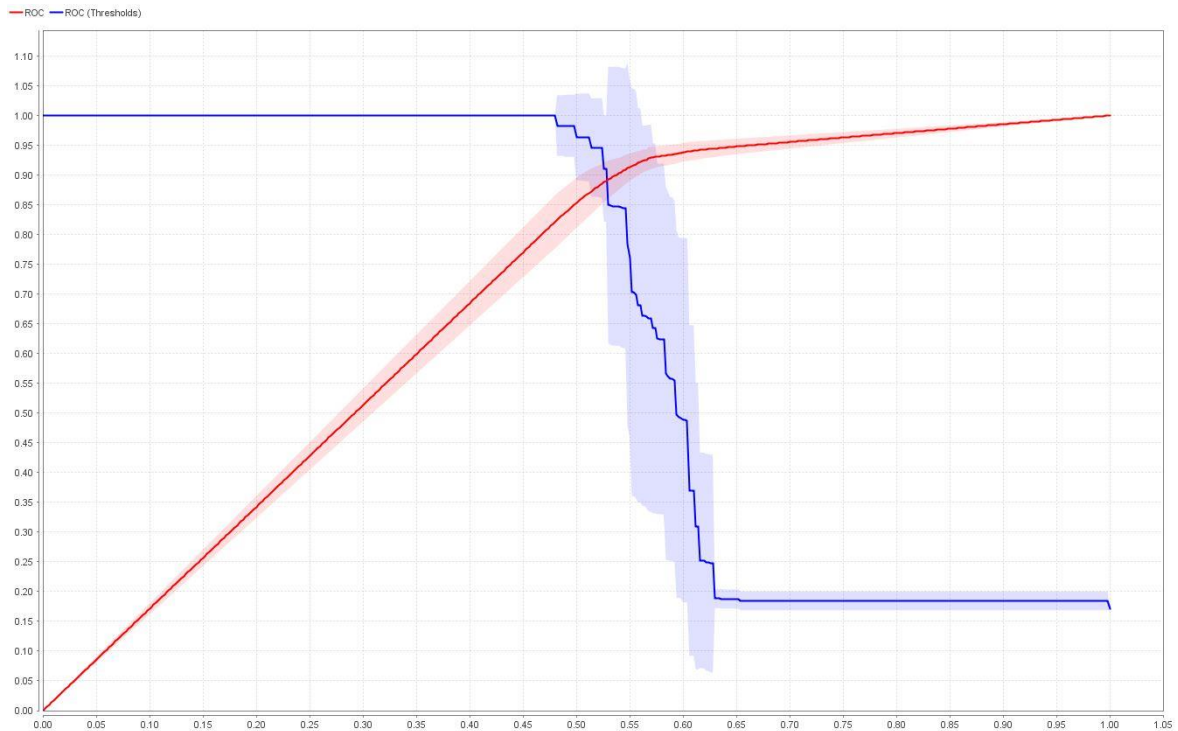


Figure 4.22 ROC curve of RF in Boosting for RapidMiner

4.12 Artificial Neural Networks

An Artificial Neural Network (ANN) is a data processing model that stimulated the human brain works. The ANN model is known as a Multilayer Perceptron (MLP) in Weka [18, 37]. The prediction accuracy is most robust and can work if training instances include errors. ANN can be separated into two main sections: Feed-backward and feed-forward neural network connections. It is a feed-forward learning algorithm and inspired by human brain neural networks. ANN models are composed of links that are the connection between nodes and multiple nodes, which are processing units and has input, hidden and output layers [1]. At the input layer nodes, layers take the input, which includes CK metrics value. Weights used for output computation and assigned on the links connected to the nodes. The fault prediction accuracy rate is output.

Some experiments and rules are available in past researches to determine the neurons number in the hidden nodes. One of the rules is that hidden layer neuron count should be 70% to 90% of the input layer size. Another rule is that in the input layer, neurons of the hidden layer number should be less than twice the neurons number [8, 10, 9]. Another critical issue for determining of hidden layers count is an activation function complexity level. Neural network complexity is increasing if hidden layers up to three [29]. If hidden layers number less than it should be, prediction accuracy decreases. If hidden layers number more than it should be, prediction results may overfit. Eventually, the optimal count of hidden layers should be determined. To achieve the best performance result, learning rate, training cycle and momentum values are critical.

- The learning rate is also called the size of the step and range between 0.0 and 1.0. The common default value of the learning rate is 0.1. It means that weights in the network are updated the 10% estimated weight error of the model with each time using the backpropagation. It controls the speed or rates the model learns. A ratio of large learning rate allows learning faster, but the small ratio of learning rate allows learning more optimal the model.
- The training cycle is the number of times repeated cycle and the default value is 500 in WEKA.
- Momentum and learning rate similar and it can improve both accuracy and training speed. Momentum range is between 0 and 1. After updating the learning rate weight, prior updates weighted average includes the weight exponentially.

20 software metrics analyzed with Neural Network graphical user interface in Figure 4.23. The most effective nine software defect prediction metrics have been selected after reviewing past researches [48]. 9 software metrics analyzed with Neural Network form in Figure 4.24.

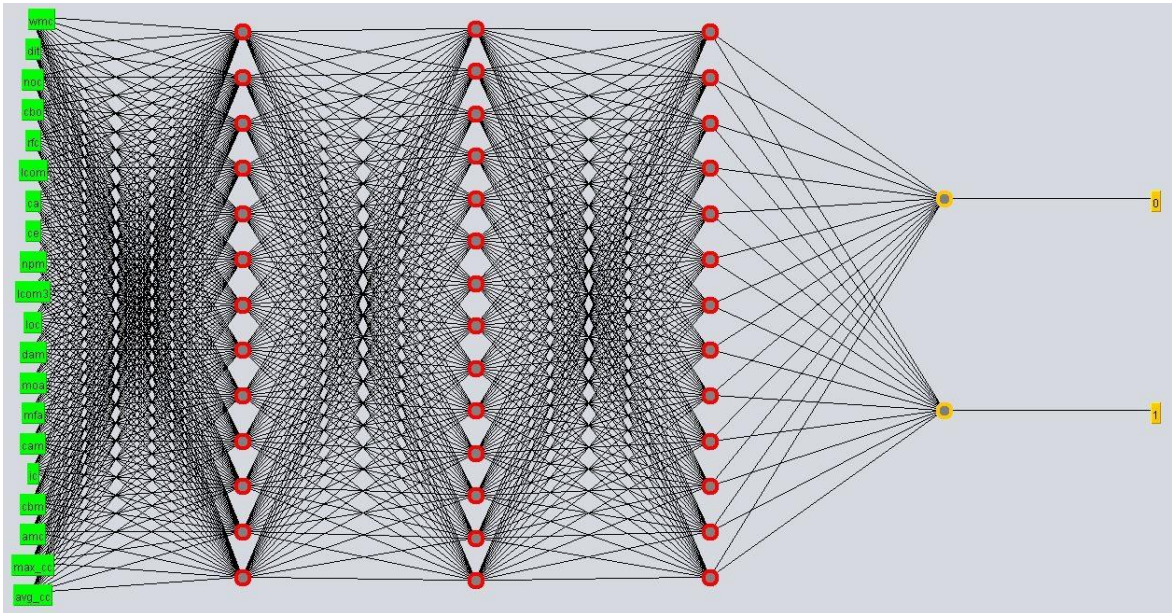


Figure 4.23 Neural Network graphical user interface form for 20 software metrics

The neural network has 20 input nodes, which are software metrics. The number of hidden layers specified as 13, 14, 13.

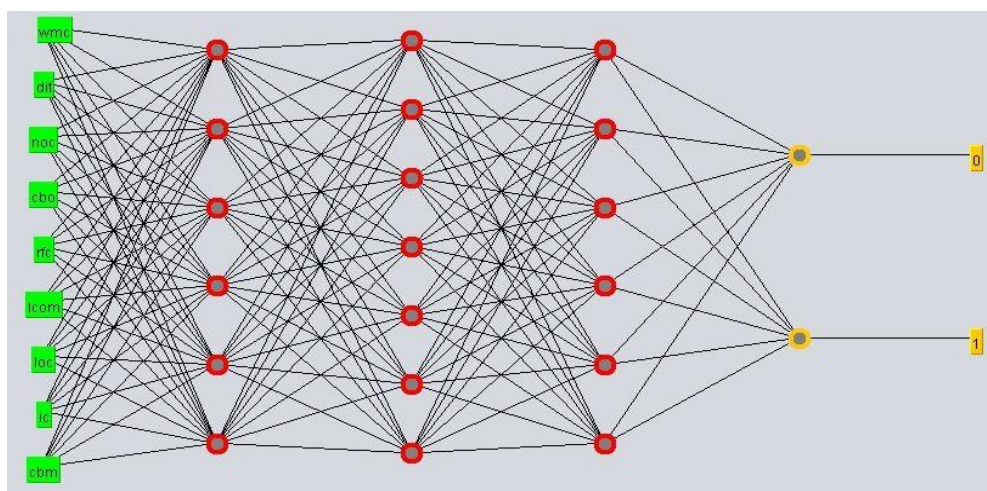


Figure 4.24 Neural Network graphical user interface form for 9 software metrics

The neural network has 9 input nodes, which are software metrics. The number of hidden layers specified as 6, 7, 6. The output layer number is 2. One of the defects the other is a non-defect binary classification. Default values in WEKA, Multilayer Perceptron using 0.2 momentum with 500 epochs and 0.3 learning rate. Default values in RapidMiner, momentum is 0.9 with 200 training cycles and 0.01 learning rate.

For comparison between RapidMiner and WEKA tools, momentum is specified as 0.2 with 500 training cycles and 0.3 learning rate in RapidMiner and there were no additional filters in both tools in Table 5.2 and Table 5.3. Screenshots of WEKA tool usage available in appendix 15 for Neural Network.

4.12.1 Artificial Neural Networks analyses in WEKA

Table 4.61 showing that 0.01 learning rate, 500 training cycle, 0.2 momentum values provide the best performance results with 13, 14, 13 hidden layers. For example, decreasing training cycles affects performance results negatively.

Table 4.61 Performance results of Training Cycle, Learning Rate, Momentum combinations of ANN in WEKA

Learning Rate	Training Cycles	Momentum	Results
0.3 (Default)	500 (Default)	0.2 (Default)	Accuracy: 70.91% AUC: 0.704 Precision: 0.700 Recall: 0.709 MAE: 0.379 RMSE: 0.447
0.3	500	0.9	Accuracy: 65.17% AUC: 0.597 Precision: 0.630 Recall: 0.652 MAE: 0.424 RMSE: 0.480
0.01	500	0.2	Accuracy: 71.12% AUC: 0.715 Precision: 0.705 Recall: 0.711 MAE: 0.388 RMSE: 0.442
0.01	500	0.9	Accuracy: 71.23% AUC: 0.709 Precision: 0.706 Recall: 0.712 MAE: 0.386 RMSE: 0.444
0.01	200	0.9	Accuracy: 70.62% AUC: 0.705 Precision: 0.698 Recall: 0.706 MAE: 0.393 RMSE: 0.446

Default settings are 0.3 learning rate, 500 training cycles and 0.2 momentum in WEKA and confusion matrix is given in Table 4.62. Improved settings, 0.01 learning rate, 500 training cycles and 0.2 momentum combination provide the best accuracy result and confusion matrix is given in Table 4.63. Default and improved settings ROC curves are given in Figure 4.25. 0.9 momentum with 500 epochs and 0.3 learning rate combinations has a less effective performance.

Table 4.62 Confusion matrix of 0.3 learning rate, 500 training cycles and 0.2 momentum combination of ANN in WEKA

Classified As	a	b
a = 0	4473	662
b = 1	1692	1266

Data are classified 71% correctly. According to the confusion matrix, 662 instances should be non-defect but classified as defective and 1692 instances should be defective but classified as non-defective.

Table 4.63 Confusion matrix of 0.01 learning rate, 500 training cycles and 0.2 momentum combination of ANN in WEKA

Classified As	a	b
a = 0	4568	567
b = 1	1770	1188

Data are classified 71% correctly. According to the confusion matrix, 567 instances should be non-defect but classified as defective and 1770 instances should be defective but classified as non-defective.

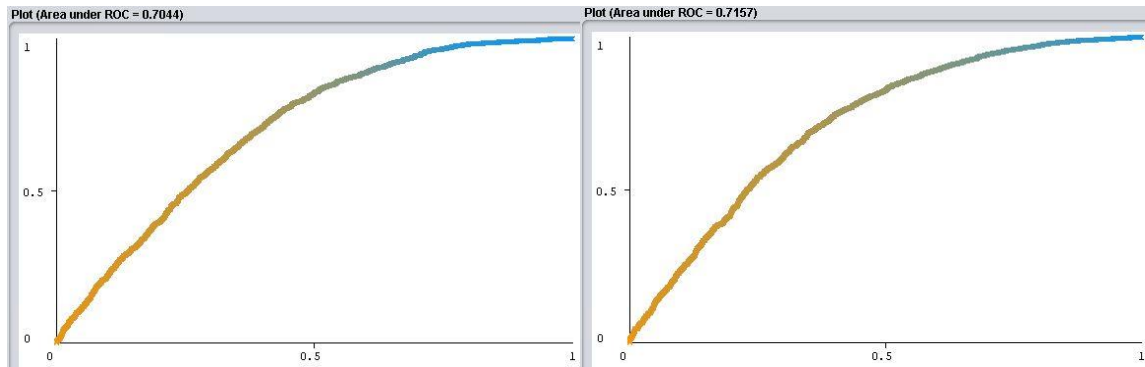


Figure 4.25 ROC curves of 0.3 learning rate, 500 training cycles, 0.2 momentum and 0.01 learning rate, 500 training cycles, 0.2 momentum combinations of ANN in WEKA

4.12.2 Artificial Neural Networks analyses in RapidMiner

Performance results of RapidMiner can be improved with different training cycles, momentum and learning rate with 13, 14, 13 hidden layers. The experiments are given in Table 4.64.

Table 4.64 Performance results of Training Cycle, Learning Rate, Momentum combinations of ANN in RapidMiner

Learning Rate	Training Cycles	Momentum	Results
0.3 (Default)	500 (Default)	0.2 (Default)	Accuracy: 71.42% AUC: 0.716 Precision: 0.726 Recall: 0.883 MAE: 0.373 RMSE: 0.444
0.3	500	0.9	Accuracy: 66.08 % AUC: 0.637 Precision: 0.668 Recall: 0.939 MAE: 0.402 RMSE: 0.488
0.3	200	0.2	Accuracy: 70.54% AUC: 0.717 Precision: Recall: 0.871 MAE: 0.382 RMSE:0.445
0.3	200	0.9	Accuracy: 66.18% AUC: 0.649 Precision: 0.681 Recall: 0.900 MAE: 0.404 RMSE: 0.481
0.01	500	0.2	Accuracy: 70.95% AUC: 0.720 Precision: 0.718 Recall: 0.892 MAE: 0.388 RMSE: 0.443
0.01	500	0.9	Accuracy: 71.25% AUC: 0.723 Precision: 0.726 Recall: 0.878 MAE: 0.384 RMSE: 0.442
0.01	200	0.2	Accuracy: 63.45% AUC: 0.624 Precision: 0.634 MAE: 0.464 RMSE: 0.481
0.01	200	0.9	Accuracy: 70.57% AUC: 0.715 Precision: 0.722 Recall: 0.873 MAE: 0.396 RMSE: 0.446

The default settings of RapidMiner is 0.3 learning rate, 500 training cycles and 0.2 momentum and confusion matrix is given in Table 4.65. Improved settings, 0.01 learning rate, 500 training cycles and 0.9 momentum combination provide the best accuracy result and confusion matrix is given in Table 4.66. ROC curve is given in Figure 4.26 for improved settings. The least effective combination is 0.2 momentum with 200 epochs and 0.01 learning rate for performance accuracy.

Table 4.65 Confusion matrix of 0.3 learning rate, 500 training cycles and 0.2 momentum combination of ANN in RapidMiner

Accuracy: 71.42%	True 1	True 0	Class Precision
Prediction 1	1246	601	67.46%
Prediction 0	1712	4534	72.59%
Class Recall	42.12%	88.30%	

Table 4.66 Confusion matrix of 0.01 learning rate, 500 training cycles and 0.9 momentum combination of ANN in RapidMiner

Accuracy: 71.25%	True 1	True 0	Class Precision
Prediction 1	1253	622	66.83%
Prediction 0	1705	4513	72.58%
Class Recall	42.36%	87.89%	

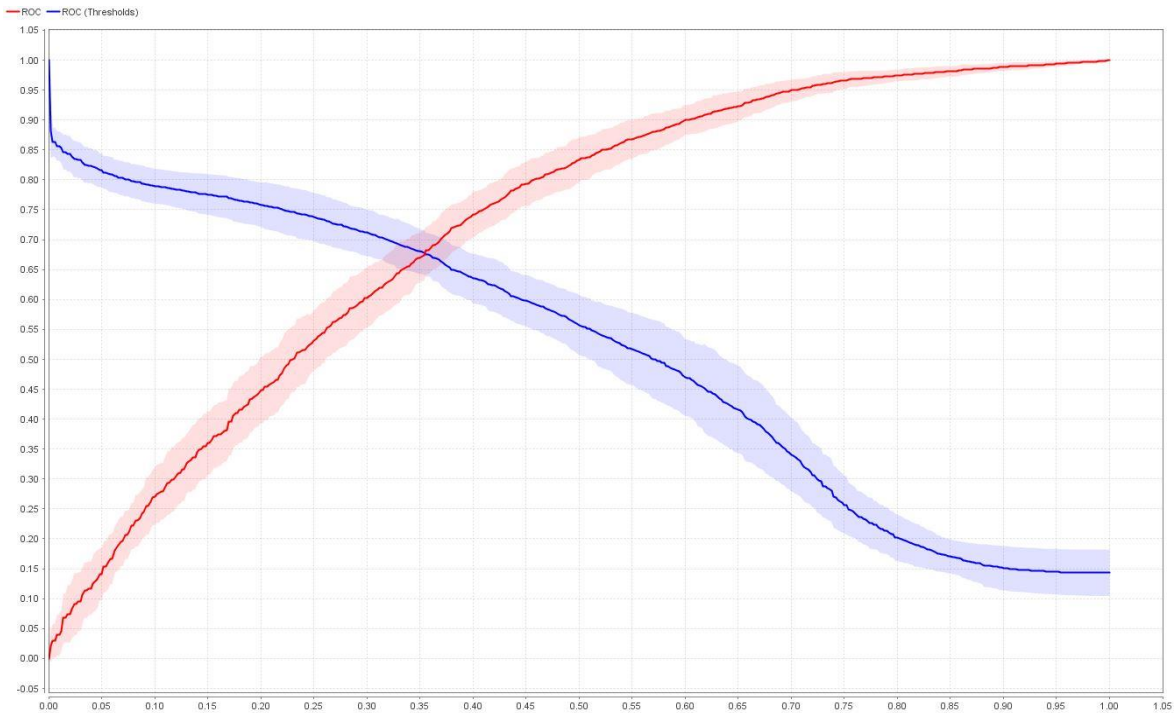


Figure 4.26 ROC curve of 0.01 learning rate, 500 training cycles, 0.9 momentum combination of ANN in RapidMiner

4.13 Nearest Neighbors

For solving problems of regression and classification, K-Nearest Neighbors (KNN) algorithm is a widely used pattern classification and supervised ML algorithm. In WEKA, Nearest neighbors are available as Ibk [37]. KNN is a sample of lazy learner algorithm which is non-parametric. The algorithm tries to find closest neighbors and data points in the training dataset aim to the classification of the new examples. It tries to solve how neighbors should be a classified problem. Usually, the Euclidean distance measure function is used to calculate distance. Other distance functions are Canberra, Chebychev, Manhattan methods [15]. The formulas of distance functions are given in Table 4.67.

- Euclidean is also called straight-line, the ordinary distance between the two data points divided by their standard deviation in Euclidean space. Euclid discusses the shortest distance and similarity between data points [4, 14].
- Canberra distance measures the sum of series differences between the feature coordinates of an object pair [35, 13].
- Chebychev also called Tchebyshev distance. It calculates the total differences distance between the pair of vectors or data points features [15].

- Manhattan distance is also known as a city block, rectilinear and taxicab. Measure the distance between the pair of data points is the sum of the absolute differences between coordinate axes [14].

Table 4.67 Distance functions formulas and parameters

Distance Functions	Formulas	Parameters
Euclidean	$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$	x, y points
Canberra	$\sum_{i=1}^d \frac{ x_i - y_i }{ x_i + y_i }$	x, y vectors
Chebychev	$\max x_{ik} - y_{jk} $	x, y points and ik, jk standart coordinates
Manhattan	$\sum_{i=1}^n x_i - y_i $	x, y points

Two different search algorithms are available in analysis. These are Linear Search and Cover Tree.

- Cover tree data structure aims to accelerate of nearest neighbor or range search. It was proved to be effective in view of space complexity. The data structure of Cover Tree can be created in $O(c^6 n \log n)$ time [5].
- There is no space complexity in the linear search. It is proved to find the right nearest neighbors but has a high estimated value. The linear search method tries to find an item sequentially. For example, it starts at the initial element. Later moves to all parts on the list until the data is found. It is called a linear search and an example of a brute-force search method. Every element visited to each other in the data structure.

‘k’ in KNN is a parameter show that the number of nearest neighbors should include. One problem is determining the best value of “k”. Because if k value is too small, it is responsive to noise in data. For enormous “k” value, consider more neighbors made it less sensitive to noise and computationally expensive. Mostly choosing the “k” value calculated as the square root of N. N mean is sample number in the training dataset. The default “k” value is 1 and the distance function is Euclidean in WEKA, but the default “k” value is 5 in RapidMiner.

For comparison between RapidMiner and WEKA tools, the “k” value is specified to 1 and Euclidean distance function is selected in RapidMiner. After a series of experiments, it was found that the “k” value should be 14 and Manhattan distance should be selected for the highest prediction accuracy for both tools. Screenshots of WEKA tool usage available in appendix 16 for Nearest Neighbors.

4.13.1 Nearest Neighbors analyses in WEKA

According to experiments, the neighbor number should be 14 with Manhattan distance function in the Linear Search algorithm in order to more staple defect prediction accuracy. Moreover, different search algorithms and distance functions were analyzed. Performance results are given in Table 4.68 using WEKA. The default settings of WEKA for the Nearest Neighbors algorithm is Euclidean. ROC curves of Euclidean and Manhattan distance function are given in Figure 4.27.

Table 4.68 Performance results of search algorithms and distance functions of KNN in WEKA

Search Algorithm	Distance Function (k=14)	Results
LinearNNSearch (Default)	Euclidean (Default)	Accuracy: 72.72% AUC: 0.745 Recall: 0.727 Precision: 0.724 MAE: 0.350 RMSE: 0.433
LinearNNSearch	Chebyshev	Accuracy: 71.61% AUC: 0.729 Recall: 0.716 Precision: 0.710 MAE: 0.364 RMSE: 0.440
LinearNNSearch	Manhattan	Accuracy: 72.72% AUC: 0.747 Recall: 0.727 Precision: 0.726 MAE: 0.346 RMSE: 0.432
Cover Tree	Euclidean	Accuracy: 72.87% AUC: 0.746 Recall: 0.729 Precision: 0.726 MAE: 0.349 RMSE: 0.433

The accuracy rate of Euclidean distance is the same as the Manhattan distance. However, Manhattan distance AUC value better than Euclidean distance. Confusion matrix of Linear search with Euclidean distance is given in Table 4.69. Confusion matrix of Linear search with Manhattan distance is given in Table 4.70.

Table 4.69 Confusion matrix of Linear search with Euclidean distance of KNN in WEKA

Classified As	a	b
a = 0	4604	531
b = 1	1676	1282

Data are classified 73% correctly. According to confusion matrix, 531 instances should be non-defect but classified as defective and 1676 instance should be defect but classified as non-defective.

Table 4.70 Confusion matrix of Linear search with Manhattan distance of KNN in WEKA

Classified As	a	b
a = 0	4661	474
b = 1	1733	1225

Data are classified 73% correctly. According to confusion matrix, 474 instances should be non-defect but classified as defective and 1733 instance should be defect but classified as non-defective.

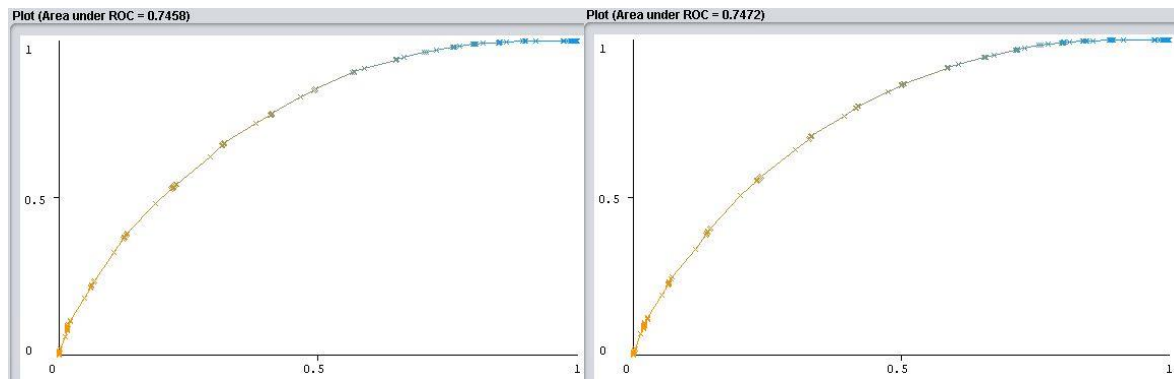


Figure 4.27 ROC curves of Euclidean with Manhattan distance of KNN in WEKA

4.13.2 Nearest Neighbors analyses in RapidMiner

The “k” value is 14 and Manhattan Distance provides the best accuracy results like WEKA. Furthermore, different search algorithms and distance functions were analyzed. Performance results are given in Table 4.71.

Table 4.71 Performance results of search algorithms and distance functions of KNN in RapidMiner

Measure	Results (k=14)
euclideanDistance (Default)	Accuracy: 68.18% AUC: 0.690 Precision: 0.705 Recall: 0.854 MAE: 0.395 RMSE: 0.459
canberraDistance	Accuracy: 50.22% AUC: 0.500 Precision: 0.684 Recall: 0.398 MAE: 0.496 RMSE: 0.525
chebychevDistance	Accuracy: 66.34% AUC: 0.675 Precision: 0.702 Recall: 0.816 MAE: 0.409 RMSE: 0.464
Manhattan Distance	Accuracy: 69.64% AUC: 0.709 Precision: 0.715 Recall: 0.867 MAE: 0.384 RMSE: 0.452

Confusion matrix of Euclidean distance is given in Table 4.72. Confusion matrix of Manhattan distance is given in Table 4.73 and ROC curve is given in Figure 4.28. Canberra distance of performance has the worst AUC value.

Table 4.72 Confusion matrix of Euclidean distance of KNN in RapidMiner

Accuracy: 68.18%	True 1	True 0	Class Precision
Prediction 1	1129	746	60.21%
Prediction 0	1829	4389	70.59%
Class Recall	38.17%	85.47%	

Table 4.73 Confusion matrix of Manhattan distance of KNN in RapidMiner

Accuracy: 69.64%	True 1	True 0	Class Precision
Prediction 1	1183	682	63.43%
Prediction 0	1775	4453	71.50%
Class Recall	39.99%	86.72%	

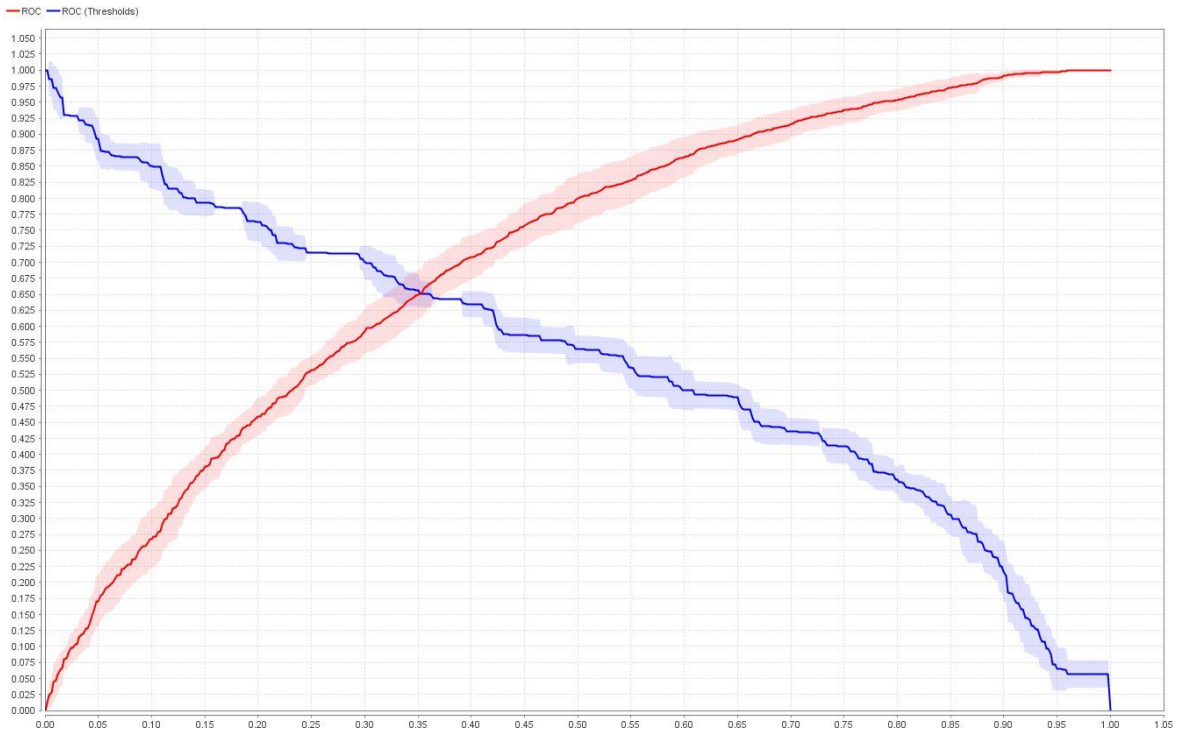


Figure 4.28 ROC curve of Manhattan distance of KNN in RapidMiner

5 RESULTS

5.1 Performance Evaluation Results

Performance evaluation parameters results are given in Table 5.1 for software defect prediction. The confusion matrix is another evaluation metrics. It shows the count of correctly classified and misclassified examples. False negative (FN) represents the negative classes that were incorrectly classified. For positive instance classified as negative. False positive (FP) represents the positive classes that were incorrectly classified. For negative instance classified as positive. True negative (TN) represents the negative classes that were correctly classified. True positive (TP) represents the positive classes that were correctly classified.

Table 5.1 Faulty and not faulty class confusion matrix

		Actual Values	
		True	False
Predicted Values	Positives	True Positives (TP)	False Positives (FP)
	Negatives	False Negatives (FN)	True Negatives (TN)

Accuracy is mostly used for evaluation analyses and binary class problem. It is the ratio of total count correct classify of values amongst the total count of predictions. It displayed in Equation 5.1.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{(\text{TP} + \text{FP} + \text{TN} + \text{FN})} \quad (5.1)$$

Precision is the ratio of predicted positive instances correctly to all the instances, which is positive. The precision calculation is in Equation 5.2.

$$\text{Precision} = \frac{\text{TP}}{(\text{TP} + \text{FP})} \quad (5.2)$$

Sensitivity (Recall) is the ratio of fault-prone classes which are correctly classified to all fault-prone classes. Recall calculation is in Equation 5.3.

$$\text{Recall} = \frac{\text{TP}}{(\text{TP} + \text{FN})} \quad (5.3)$$

The Area Under the Curve (AUC) means a characteristic ROC. AUC represented a two-dimensional graph with FP instances on the x-axis and TP instances on the y-axis [26]. For unbalanced and noisy data, AUC is an effective method. AUC is the most effective and reliable technique for the evaluation of performance and classification algorithms. MAE measure all absolute errors average and differences between predicted and original instance. The lower of the MAE value means better performance for prediction.

Another performance indicator is the RMSE. It compares observed and the predicted value by a model. The small value of RMSE indicates that predicted values are close. Performance results of 11 ML techniques and 2 SVM libraries are given in Table 5.2. and 5.4 for Weka. For the RapidMiner tool, 9 ML techniques and 2 SVM libraries are given in Table 5.3 and Table 5.5. Because Bayesian Network and Part techniques are not available, both default and improved results are represented. Thus, the effectiveness of improvements can be evaluated.

Table 5.2 Performance results of reliability prediction by default values of WEKA

Technique	Accuracy	AUC	Precision	Recall	MAE	RMSE
J48	72.25%	0.694	0.715	0.722	0.325	0.480
RF	76.94%	0.812	0.766	0.769	0.321	0.402
Naïve Bayes	67.23%	0.676	0.659	0.672	0.329	0.554
Bayesian Network	71.51%	0.733	0.707	0.715	0.350	0.443
Part	72.55%	0.748	0.718	0.726	0.341	0.434
KNN	72.72%	0.745	0.724	0.727	0.350	0.433
SMO	68.04%	0.575	0.705	0.680	0.319	0.565
LibSVM	67.66%	0.563	0.728	0.677	0.323	0.568
LibLinear	64.09%	0.563	0.614	0.614	0.359	0.599
ANN	70.91%	0.704	0.700	0.709	0.379	0.447
Bagging	75.22%	0.786	0.749	0.752	0.335	0.413
AdaBoost	66.52%	0.653	0.669	0.665	0.438	0.464
LR	69.94%	0.712	0.696	0.699	0.402	0.448

Performance results were analyzed with the default values of WEKA in Table 5.2. The default value of iteration is 100 in RF and the classifier is REPTree in Bagging. The default value of kernel type is Radial Basis in LibSVM and L2-regularized L2-loss support vector classification in LibLinear. Performance results show that RF and Bagging techniques have the highest AUC value and accuracy rate. However, LibSVM and LibLinear have the worst AUC value. These techniques are the least effective models for predicting software reliability for the default values of LibSVM and LibLinear in WEKA. The AUC value of LibSVM and LibLinear same, but the accuracy rate of LibLinear worse than LibSVM.

Table 5.3 Performance results of reliability prediction by default values of RapidMiner

Technique	Accuracy	AUC	Precision	Recall	MAE	RMSE
J48	72.94%	0.730	0.738	0.886	0.346	0.440
RF	74.46%	0.792	0.753	0.889	0.335	0.412
Naïve Bayes	67.07%	0.678	0.682	0.899	0.329	0.555
KNN	68.18%	0.690	0.705	0.854	0.395	0.459
SMO	64.88%	0.666	0.669	0.900	0.423	0.477
LibSVM	66.91%	0.674	0.678	0.910	0.422	0.463
LibLinear	68.97%	0.692	0.683	0.954	0.405	0.456
ANN	71.42%	0.716	0.726	0.883	0.373	0.444
Bagging	73.14%	0.746	0.726	0.926	0.363	0.429
AdaBoost	67.66%	0.655	0.675	0.943	0.413	0.461
LR	69.86%	0.711	0.702	0.910	0.403	0.448

Performance results were analyzed with the default values of RapidMiner in Table 5.3. The default value of iteration is 100 with gain ratio criterion in RF and the classifier is Decision Tree with information gain in Bagging. The default value of kernel type is Polynomial in SMO. The default value of the classifier is Decision Stump in AdaBoost. Performance results show that RF and Bagging techniques have the highest AUC value and accuracy rate. However, AdaBoost and SMO have the worst AUC value. These techniques are the least effective models for predicting the reliability of software for the default values of RapidMiner.

Table 5.4 Performance results of reliability prediction by improved values of WEKA

Technique	Accuracy	AUC	Precision	Recall	MAE	RMSE
J48	73.13%	0.724	0.725	0.731	0.340	0.453
RF	77.07%	0.816	0.768	0.771	0.320	0.400
Naïve Bayes	68.36%	0.686	0.670	0.684	0.326	0.517
Bayesian Network	72.54%	0.738	0.719	0.725	0.348	0.437
Part	72.55%	0.748	0.718	0.726	0.341	0.434
KNN	72.72%	0.747	0.726	0.727	0.346	0.432
SMO	73.18%	0.661	0.736	0.732	0.268	0.517
LibSVM	68.57%	0.607	0.676	0.686	0.314	0.560
LibLinear	69.87%	0.620	0.695	0.699	0.301	0.548
ANN	71.12%	0.715	0.705	0.711	0.388	0.442
Bagging	76.59%	0.811	0.764	0.766	0.329	0.402
AdaBoost	76.62%	0.781	0.763	0.766	0.238	0.471
LR	69.94%	0.712	0.696	0.699	0.402	0.448

Performance results were analyzed with improved values of WEKA in Table 5.4. The improved value of iteration is 500 in RF and the classifier is RF in Bagging. The improved value of kernel type is Linear in LibSVM and L1-regularized logistic regression in LibLinear. Performance results show that RF and Bagging techniques have the highest AUC value. However, LibSVM and LibLinear have the worst AUC value. These techniques are the least effective models for predicting the reliability of software for improved values of WEKA.

Table 5.5 Performance results of reliability prediction by improved values of RapidMiner

Technique	Accuracy	AUC	Precision	Recall	MAE	RMSE
J48	72.82%	0.735	0.737	0.887	0.351	0.437
RF	76.62%	0.815	0.772	0.894	0.324	0.401
Naïve Bayes	67.81%	0.688	0.691	0.890	0.327	0.527
KNN	69.64%	0.709	0.715	0.867	0.384	0.452
SMO	68.82%	0.691	0.681	0.953	0.405	0.456
LibSVM	66.91%	0.674	0.678	0.910	0.422	0.463
LibLinear	68.97%	0.692	0.683	0.954	0.405	0.456
ANN	71.25%	0.723	0.726	0.878	0.384	0.442
Bagging	74.66%	0.790	0.738	0.930	0.361	0.415
AdaBoost	74.46%	0.693	0.738	0.925	0.339	0.443
LR	69.86%	0.711	0.702	0.910	0.403	0.448

Performance results were analyzed with improved values of RapidMiner in Table 5.5. The improved value of iteration is 500 with information gain in RF and the classifier is RF with information gain in Bagging. The improved value of kernel type is Radial Basis in LibSVM. Performance results show that RF and Bagging techniques have the highest AUC value. However, LibSVM and LibLinear have the worst AUC value. These techniques are the least effective models for predicting the reliability of software for improved values of RapidMiner.

Table 5.6 AUC results of WEKA and RapidMiner by improved values

Technique	WEKA AUC	RapidMiner AUC
J48	0.724	0.735
RF	0.816	0.815
Naïve Bayes	0.686	0.688
Bayesian Network	0.738	-
Part	0.748	-
KNN	0.747	0.709
SMO	0.661	0.691
LibSVM	0.607	0.674
LibLinear	0.620	0.692
ANN	0.715	0.723
Bagging	0.811	0.790
AdaBoost	0.781	0.693
LR	0.712	0.711

Bayesian network and Part is not available in RapidMiner. RF has the highest AUC result for WEKA and RapidMiner tool. According to Table 5.6, AUC results of RF are nearly same for WEKA and RapidMiner tool. The AUC value of WEKA is 0.816 and RapidMiner is 0.815 for RF. There are no significant differences between Naïve Bayes AUC results of WEKA and RapidMiner. Moreover, AUC results of LR are nearly same for WEKA and RapidMiner.

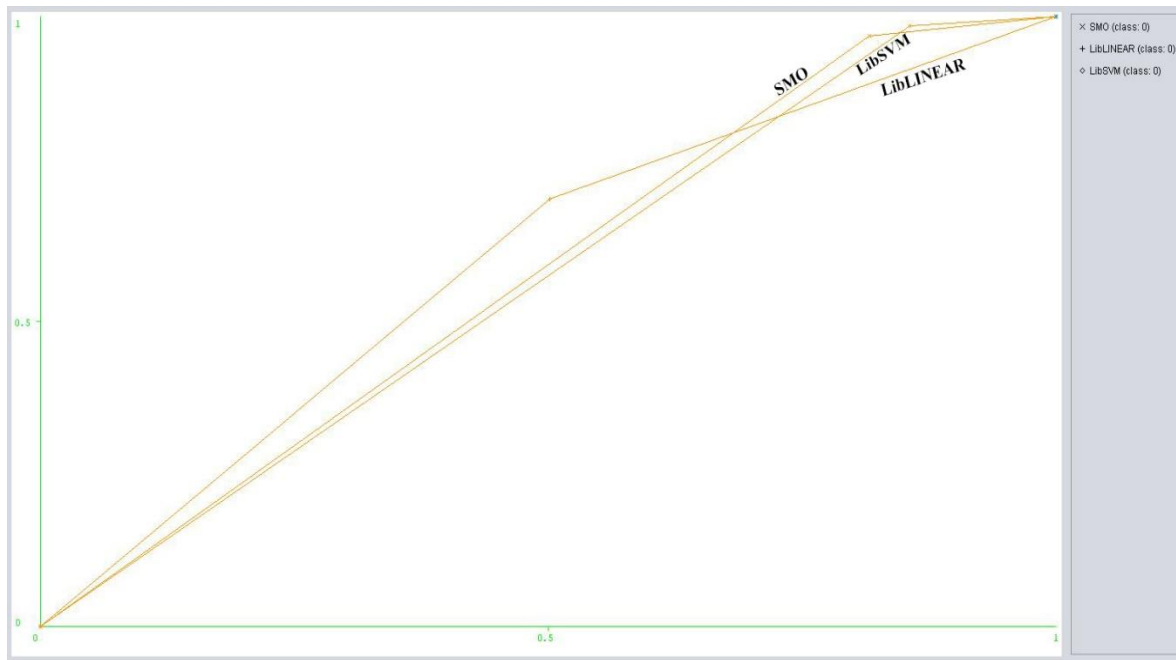


Figure 5.1 Default configuration of SVMs ROC curves

ROC curves comparisons of SVMs such as SMO, LibLINEAR and LibSVM are given in Figure 5.1 for default configuration in WEKA. The y-axis presents the true positive rate. The x-axis presents false positive rate. SMO uses polynomial, LibSVM uses Radial Basis kernel type and LibLINEAR uses L2-regularized L2-loss support vector classification (dual) SVM type. AUC performance results are 0.575 for SMO, 0.563 for LibLINEAR and LibSVM. LibLINEAR and LibSVM have the same AUC result, but the accuracy rate of LibSVM higher than LibLINEAR. It is 67%.

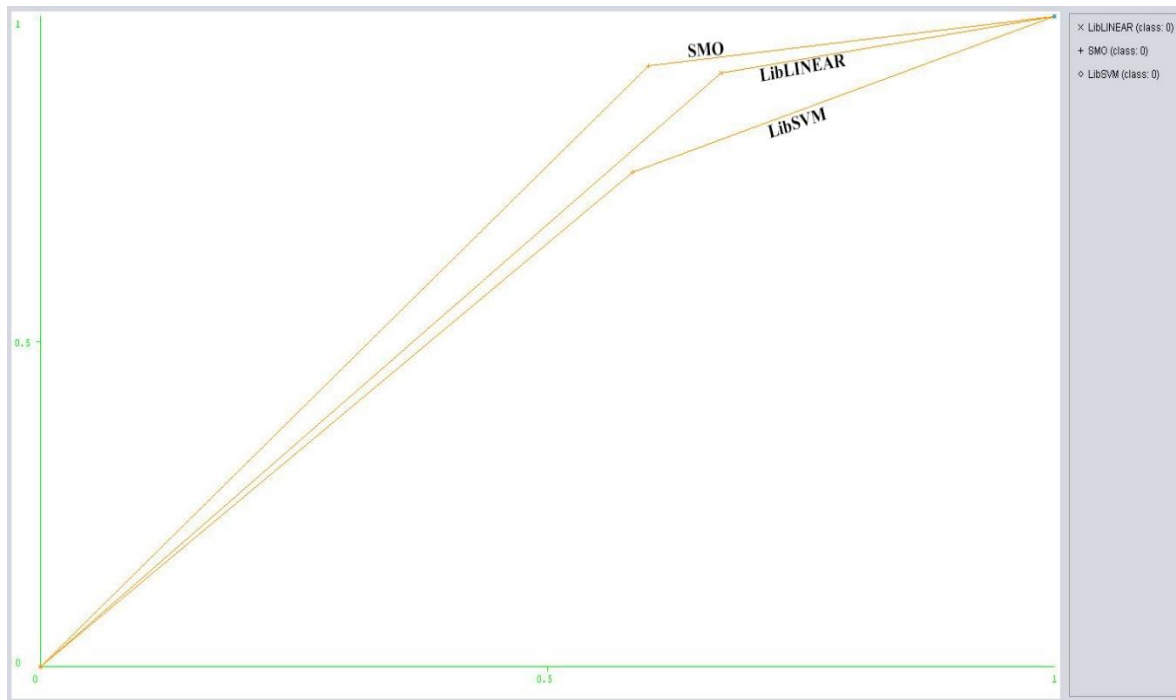


Figure 5.2 Improved configuration of SVMs ROC curves

ROC curves comparison of SMO, LibLINEAR and LibSVM are given in Figure 5.2 for improved configuration in WEKA. The y-axis presents the true positive rate. The x-axis presents false positive rate. SMO uses PUK, LibSVM uses linear kernel type and LibLINEAR uses L1-regularized logistic regression SVM type. AUC performance results are 0.661 for SMO, 0.620 for LibLINEAR, 0.607 for LibSVM.

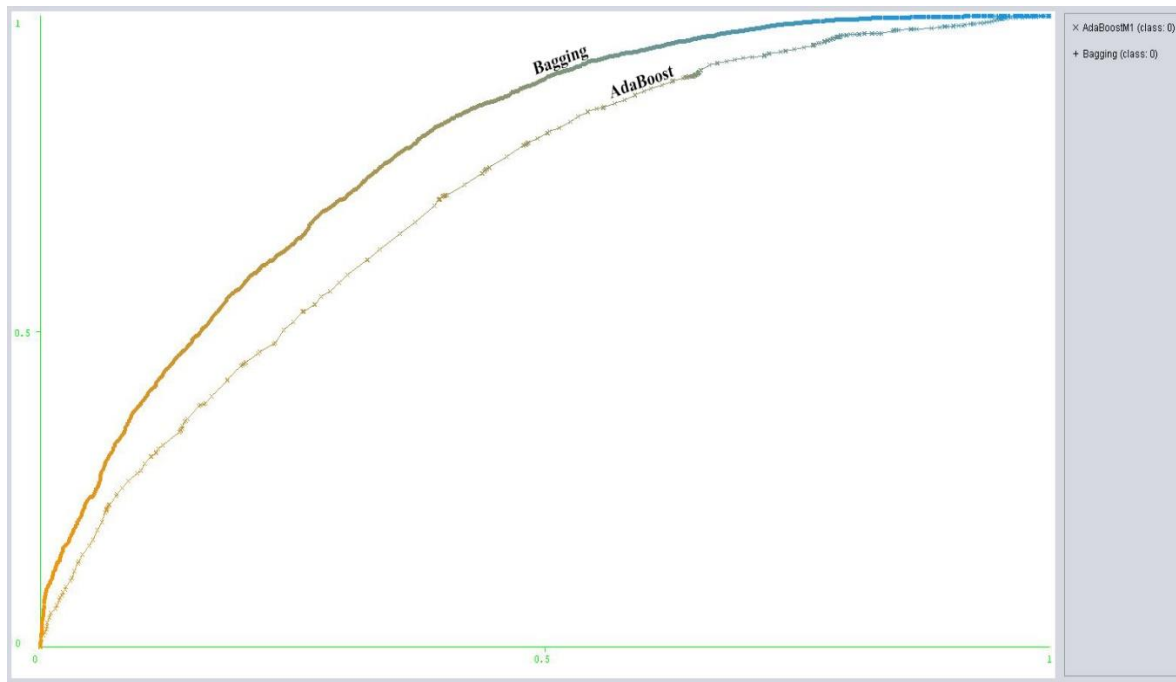


Figure 5.3 Default configuration of Bagging and Boosting ROC curves

ROC curves comparison of Bagging and Boosting are given in Figure 5.3 for default configuration in WEKA. The y-axis presents the true positive rate. The x-axis presents false positive rate. Bagging uses REPTree, AdaBoost uses the Decision Stump classifier. AUC performance results are 0.786 for Bagging, 0.653 for AdaBoost.

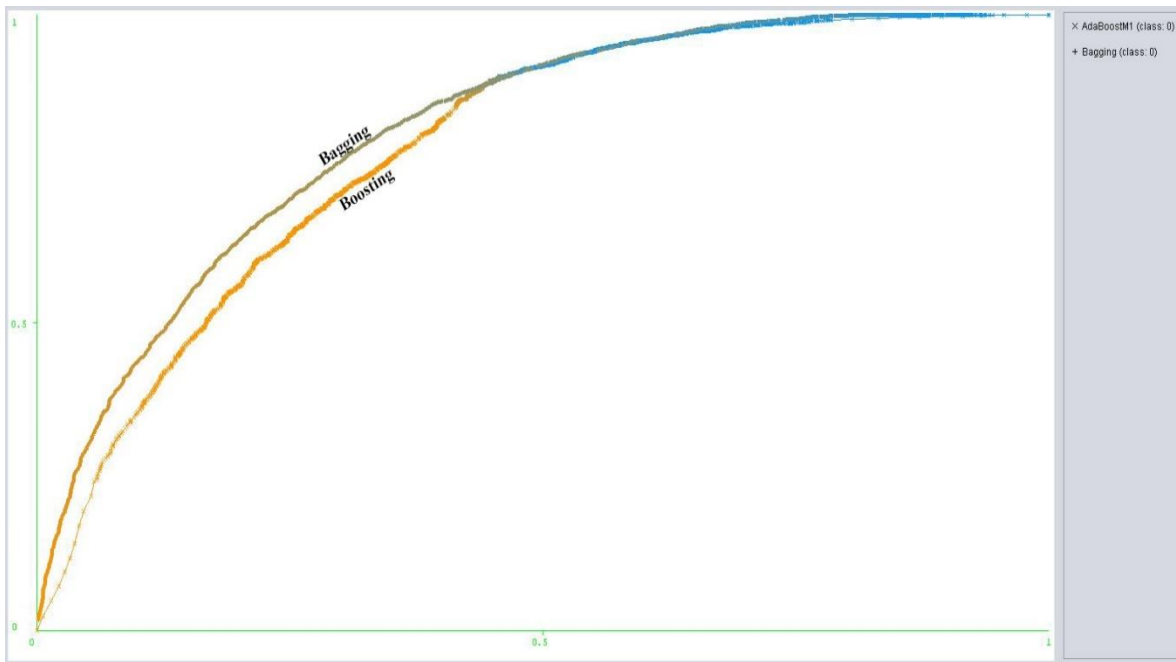


Figure 5.4 Improved configuration of Bagging and Boosting ROC curves

ROC curves comparison of Bagging and Boosting are given in Figure 5.4 for improved configuration in WEKA. The y-axis presents the true positive rate. The x-axis presents false positive rate. Bagging and AdaBoost use the Random Forest classifier. AUC performance results are 0.811 for Bagging, 0.781 for Boosting.

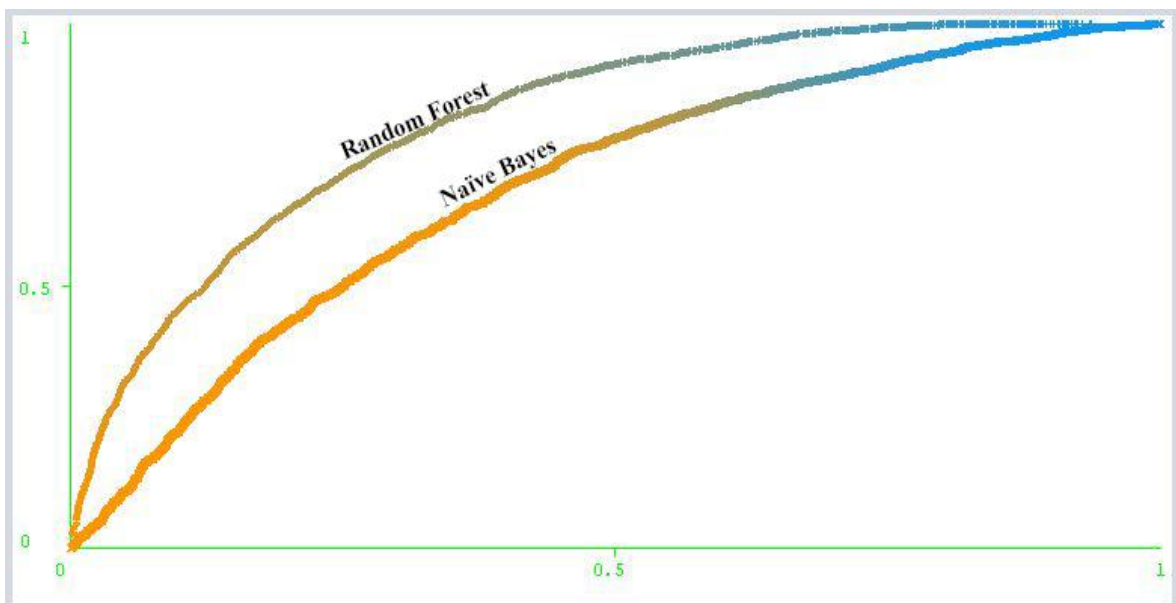


Figure 5.5 Improved configuration of RF and Naïve Bayes ROC curves

ROC curves comparison of RF and Naïve Bayes are given in Figure 5.5 for improved configuration in WEKA. The y-axis presents the true positive rate. The x-axis presents false positive rate. RF iteration count is 500. Naïve Bayes use kernel. AUC performance results are 0.816 for RF, 0.686 for Naïve Bayes.

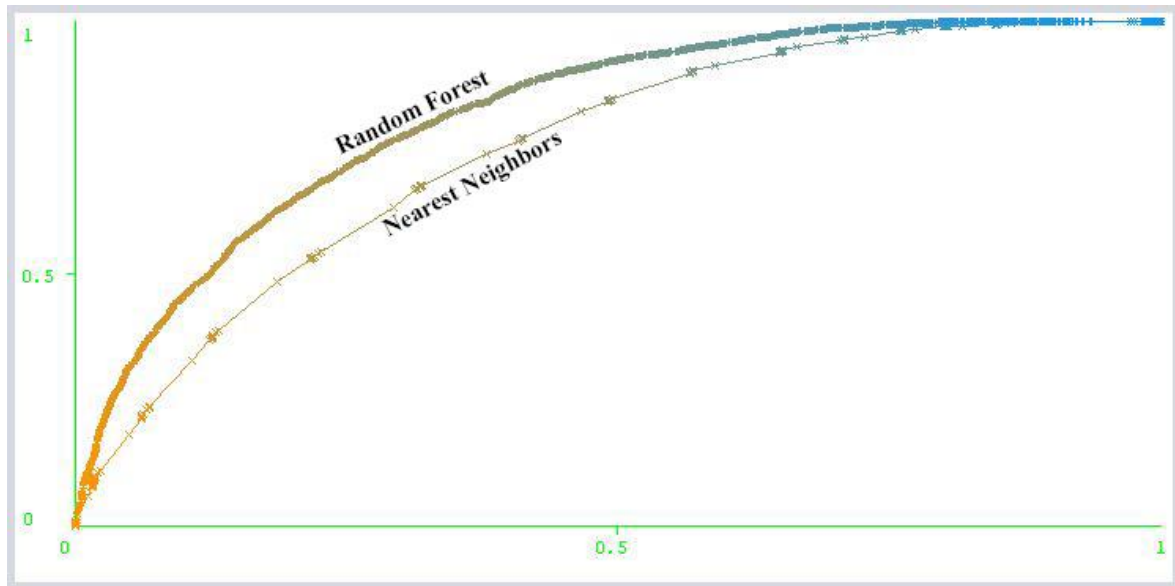


Figure 5.6 Improved configuration of RF and Nearest Neighbors ROC curves

ROC curves comparison of RF and KNN are given in Figure 5.6 for improved configuration in WEKA. The y-axis presents the true positive rate. The x-axis presents false positive rate. RF iteration count is 500. “k” value is 14 for KNN. AUC performance results are 0.816 for RF, 0.747 for KNN.

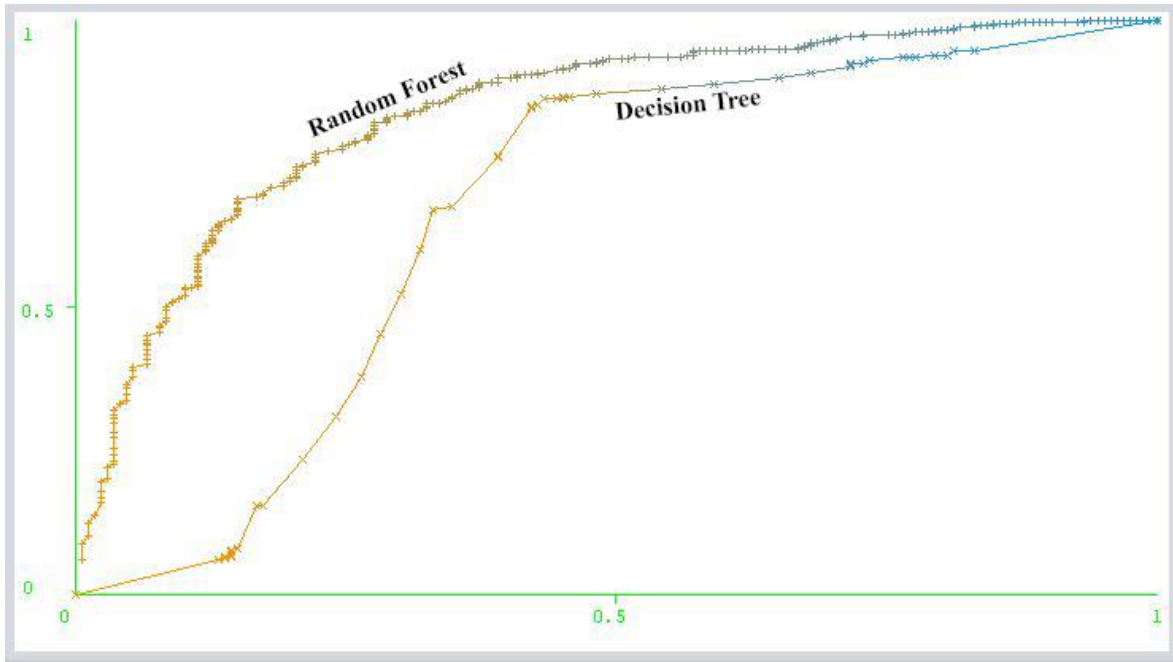


Figure 5.7 Improved configuration of RF and Decision Tree ROC curves

ROC curves comparison of RF and Decision Tree are given in Figure 5.7 for Ant software. The version of Ant is 1.7 and include 745 instance, 166 defective modules. The y-axis presents the true positive rate. The x-axis presents false positive rate. AUC performance results are 0.816 for RF, 0.724 for Decision Tree.

5.2 Findings

According to Bayesian Network, from the results, it is clear that the WMC software metric was found significant predictor for both 9 and 20 software prediction metrics. Other significant predictors are NOC, CBO, LCOM, NPM, CAM, DIT and RFC software metrics for defect prediction. Tertiary software metrics are CA, CE, LCOM3, MFA and LOC. Fourthly software metrics are DAM, MOA, IC, AMC and MAX_CC. The least effective software metrics are CBM and AVG_CC. However, results changed for most used 9 software metrics. For 9 CK metrics, WMC is still a major attribute followed by NOC, CBO, RFC, LCOM and DIT. Tertiary software metrics are LOC and IC. The least useful software metric is CBM. These findings of this study could be included significant effective results for the reliability issue. From these results, it is clear that more reliable software and less defect classes in software systems may provide.

Values of AUC and accuracy rate were obtained by the ML techniques performance results and evaluated with each other to comparing performance measurement. Performance results show that RF and Bagging techniques have the highest and effective accuracy rate and AUC values for both default and improved results, according to RapidMiner and WEKA ML tools. RF provides the highest AUC value in RapidMiner and WEKA performance results. AUC is 0.815 for RapidMiner and 0.816 for WEKA. After the improvement of ML techniques, still, RF and Bagging have the highest accuracy and AUC value. Followed by these ML techniques are Boosting and Rule-based classifications. Both of them have excellent accuracy and AUC value according to improved performance results of WEKA. However, the AUC value of Rule-based and Nearest Neighbors nearly the same. According to improved performance results of RapidMiner, Decision Tree and Neural Network techniques have good accuracy and AUC value after RF and Bagging techniques. After the improvement of the default configuration of WEKA, Neural Network techniques become better than Decision Tree. Also, Boosting algorithm performance results is increasing. After the improvement of the default configuration of RapidMiner, Nearest Neighbor and Boosting techniques become better than Naïve Bayes's performance results. Nevertheless, the AUC value of Nearest Neighbor and LR nearly the same.

Regarding accuracy rate and AUC performance results, there are no essential differences between Rule-based classification and Nearest Neighbors for improved results of WEKA. LibSVM has the worst accuracy rate and AUC value for both tools. SMO and LibLinear performance are better than LibSVM for both tools. However, this analysis found evidence for SMO, LibSVM and LibLinear techniques are the least effective models for predicting defect-prone modules. Another finding is that the AUC value of SMO and LibLINEAR nearly the same for the improved result of RapidMiner.

The results demonstrated that from the best ML techniques to the worst for software defect predictions are RF, Bagging, AdaBoosting, Rule-based classification, Naïve Bayes, Bayesian Network, Decision Tree, Neural Network, LR, SMO, LibLinear and LibSVM for improved performance results of WEKA. For improved performance results of RapidMiner tool, RF, Bagging, Decision Tree, Neural Network, LR, Neural Network, Boosting, LibLinear, SMO, Naïve Bayes, LibSVM from the best ML techniques to the worst.

6 CONCLUSION

This thesis argues predicting and improving software reliability by comparing the performance of various 11 ML techniques and 2 SVM libraries with tenfold cross-validation and OO software metrics. Overall, the results demonstrate a significant effect of ML techniques and OO software metrics. Besides, these results provide additional information about the significant attributes of software metrics. For 20 software metrics, WMC is a major attribute followed by NOC, CBO, LCOM, NPM, CAM, DIT and RFC. The results provide evidence for which software metric is more important for software reliability.

Moreover, these findings provide additional information about ML tools, which are WEKA and RapidMiner, but results may change according to the analyzed tool. The main conclusion shows and confirms that RF and Bagging are important contributors to software reliability and useful for fault prediction. SMO, LibSVM, LibLINEAR SVMs are the least effective model for defect classification. The main contribution of these analyses to evaluate the performance of ML algorithms with a large OO software dataset. Another contribution of this study is to display software metrics effectiveness for the prediction of a software reliability and quality.

The answers to research questions are WMC is the most and CBM and AVG_CC's least effective software metrics to determine software reliability. Random Forest is the most effective ML techniques to estimate software defect prediction.

In future studies, the number of OO software projects should be increased. It is also planned to apply new ML algorithms with various kernel types, confidence factors, iterations, search algorithms, classifiers, estimators, and different software metrics. The quality of defect prediction standards depends on the choice of dataset and ML techniques. Future investigations are necessary to validate performance conclusions. The performance results provide a good starting point for discussion and further research.

REFERENCES

- [1] A. K. Jain, J. Mao and K. M. Mohiuddin, “Artificial neural networks: A tutorial,” *IEEE Comput.*, pp. 31–44, Mar., 1996.
- [2] A. Okutan and O. Yildiz, “Software defect prediction using bayesian networks,” *Empirical Software Engineering*, pp. 1–28, 2012.
- [3] A. S. Galathiya, A. P. Ganatra and C. K. Bhensdadia, “Improved Decision Tree Induction Algorithm with Feature Selection, Cross Validation, Model Complexity and Reduced Error Pruning,” (*IJCSIT*) *International Journal of Computer Science and Information Technologies*, Vol. 3 (2) , 2012.
- [4] Akarsh Goyal, Neel Sheth, N Sujith Kumar Reddy, “Software Defect Prediction using Euclidean distance probability,” *International Journal of Soft Computing* 11(3):203-206, 2016.
- [5] Alina Beygelzimer, Sham Kakade, John Langford, “Cover Trees for Nearest Neighbor,” Appearing in *Proceedings of the 23rd International Conference on Machine Learning*, Pittsburgh, 2006.
- [6] Arisholm, E., Briand, L.C., Johannessen, “A Systematic and Comprehensive Investigation of Methods to Build and Evaluate Fault Prediction Models,” *Journal of System and Software*, 83, 1, 2–17, 2010.
- [7] Bansiya, J. and Davis, “A Hierarchical Model for Quality Assessment of Object-Oriented Designs,” *IEEE Transactions on Software Engineering*, Vol. 28, No.1, pp.4-17, 2002.
- [8] Berry, M.J.A. and Linoff, “*Data Mining Techniques*,” NY: John Wiley & Sons, 1997
- [9] Blum, “*Neural Networks in C++*,” NY: Wiley, 1992.
- [10] Boger, Z., and Guterman, “Knowledge extraction from artificial neural network models,” *IEEE Systems, Man and Cybernetics Conference*, Orlando, FL, USA, 1997.

- [11] Chih-Jen Lin, S. Sathiya Keerthi, "Asymptotic Behaviors of Support Vector Machines with Gaussian Kernel," *Neural Computation*, Volume: 15, Issue: 7, 2003.
- [12] D. P. Tegarden, S. D. Sheetz, and D. E. Monarchi, "A software complexity model of object-oriented systems," *Decision Support Systems*, vol. 13, no. 3-4, pp. 241–262, 1995.
- [13] Deepinder Kaur, "A Comparative Study of Various Distance Measures for Software fault prediction," *International Journal of Computer Trends and Technology (IJCTT)*, volume 17 Issue 3, 2014.
- [14] Derya Birant, Elife Öztürk Kıyak, Kökten Ulaş Birant, "An Ordinal Classification Approach for Software Bug Prediction," 2019.
- [15] Dinesh Kumar, Jitender Kumar Chhabra, Vijay Kumar, "Performance Evaluation of Distance Metrics in the Clustering Algorithms," *INFOCOMP*, v. 13, no. 1, p. 38-51, 2014.
- [16] F. B. Abreu, "Design metrics for object-oriented software systems," *ECOOP'95 quantitative methods workshop*, Aarhus, 1995.
- [17] G. Boetticher, T. Menzies and T. J. Ostrand, "Promise repository of empirical software engineering data," 2007. [Online]. Available: <http://promisedata.org/repository>.
- [18] G. Holmes, A. Donkin, and I. Witten, "Weka: A machine learning workbench," in *Proc. 2nd Aust. New Zealand Conf. Intell. Inf.Syst.*, pp. 1269–1277, 1994.
- [19] Genero M., "Defining and Validating Metrics for Conceptual Models," Ph.D. thesis, University of Castilla-La Mancha, 2002.
- [20] Haidar Osman, Mohammad Ghafari, Oscar Nierstrasz, "Hyperparameter Optimization to Improve Bug Prediction Accuracy," *IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation*, 2017.

- [21] Haijin JI, Song HUANG, Xuewei LV, Yaning WU, Nonmembers and Yuntian FENG, “Empirical Studies of a Kernel Density Estimation Based Naive Bayes Method for Software Defect Prediction,” *IEICE Transactions* 102-D(1): 75-84, 2019.
- [22] Harrison R., Counsell S. and Nithi R., “Coupling Metrics for Object-Oriented Design,” *5th International Software Metrics Symposium Metrics*, pp. 150-156, 1998.
- [23] Hsu, Chan and Lin, “A Practical Guide to support vector classification,” 2010.
- [24] I. Gondra, “Applying machine learning to software fault proneness prediction,” *Journal of Systems and Software*, vol. 81, no. 2, pp. 186–195, 2008.
- [25] I. H. Laradji, M. Alshayeb, and L. Ghouti, “Software defect prediction using ensemble learning on selected features,” *Information and Software Technology*, 58:388– 402, 2015.
- [26] K. Huang, “Discriminative Naive Bayesian Classifiers,” Department of Computer Science and Engineering, the Chinese University of Hong Kong, 2003.
- [27] K. O. Elish and M. O. Elish, “Predicting defect-prone software modules using support vector machines,” *J. Syst. Softw.*, vol. 81, no. 5, pp. 649–660, 2008.
- [28] K. P. Murphy, “Naive Bayes Classifiers,” Technical Report, October, 2006.
- [29] Karsoliya, “Approximating Number of Hidden layer neurons in Multiple Hidden Layer BPNN Architecture,” *International Journal of Engineering Trends and Technology*, 2012.
- [30] Kim E. M, Chang O. B, Kusumoto S, Kikuno T., “Analysis of metrics for object-oriented program complexity,” *Computer Software and Applications Conference, IEEE Computer*, pp. 201-207, 1994.
- [31] L. Breiman, “Random Forests,” *Machine Learning*, 2001, vol. 45, no. 1, pp. 5-32, 2001.

- [32] L. Briand, J. Daly, V. Porter, and J. Wust, "Exploring the relationships between design measures and software quality," *Journal of Systems and Software*, vol. 5, pp. 245-273, 2000.
- [33] L. Brieman, "Bagging Predictors," *Machine Learning*, vol. 24, pp. 123-140, 1996.
- [34] L. Etzkorn, J. Bansiya, and C. Davis, "Design and code complexity metrics for OO classes," *Journal of Object-Oriented Programming*, vol. 12, no. 1, pp. 35–40, 1999.
- [35] Lance, G. N. and Williams, "Computer programs for hierarchical polythetic classification (similarity analyses)," *Computer*, 9(1):60–64, 1966.
- [36] M. H. Halstead, "Elements of Software Science," Elsevier Science, New York, NY, USA, 1997.
- [37] M. Hal et al., "The WEKA data mining software: An update," *SIGKDD Explor.*, vol. 11, no. 1, pp. 10-18, 2009.
- [38] M. Lorenz and J. Kidd, "Object-Oriented Software Metrics," Prentice Hall, Englewood, NJ, USA, 1994.
- [39] M.-H. Tang, M.-H. Kao, and M.-H. Chen, "Empirical study on object-oriented metrics," in *Proceedings of the 6th International Software Metrics Symposium*, pp. 242–249, November, 1999.
- [40] M.M.T. Thwin, T.-S. Quah, "Application of Neural Networks for Software Quality Prediction Using Object-Oriented Metrics," *Proc. IEEE Int'l Conf. Software Maintenance (ICSM)*, 2003.
- [41] Malhotra, Shukla and Sawhney, "Assessment of Defect Prediction Models Using Machine Learning Techniques for Object-Oriented Systems," *5th International Conference on Reliability, Infocom Technologies and Optimization*, 2016.
- [42] P. Yu, T. Systa and H. Muller, "Predicting Fault-Proneness Using OO Metrics: An Industrial Case Study," *Proc. Sixth European Conf. Software Maintenance and Reeng. (CSMR 2002)*, pp. 99-107, Mar., 2002.

- [43] Pai, G.J., Dugan, J.B., “Empirical analysis of software fault content and fault proneness using Bayesian methods,” *IEEE Transactions on Software Engineering*, 33, 10, 675–686, 2007.
- [44] R. Martin, “OO design quality metrics—an analysis of dependencies,” in *Proceedings of the Workshop Pragmatic and Theoretical Directions in Object-Oriented Software Metrics*, 1994.
- [45] Rajanikanth Aluvalu, “A Reduced Error Pruning Technique for Improving Accuracy of Decision Tree Learning,” *International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-3, Issue-5, June, 2014.*
- [46] RapidMiner Open Source Predictive Analytics Platform Available: <https://rapidminer.com/get-started/>.
- [47] S. R. Chidamber and C. F. Kemerer, “A Metrics Suite for Object Oriented Design,” *IEEE Transactions on Software Engineering*, 20(6), pp. 476-493, 1994.
- [48] Sandeep Reddivari and Jayalakshmi Raman, “Software Quality Prediction: An Investigation based on Machine Learning,” *IEEE 20th International Conference on Information Reuse and Integration for Data Science*, 2019.
- [49] SharbleR. C. and Cohen S., “The object-oriented brewery: A comparison of two object oriented development methods,” *ACM SIGSOFT Software Engineering Notes*, 18, 2, pp.60-73, 1993.
- [50] T. Gyimothy, R. Ferenc, and I. Siket, “Empirical validation of object-oriented metrics on open source software for fault prediction,” *Software Engineering, IEEE Transactions on*, vol.31, no. 10, pp. 897 – 910, Oct, 2005.
- [51] T. J. McCabe, “A Complexity Measure,” *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308–320, 1976.
- [52] T. Zimmermann, R. Premraj and A. Zeller, “Predicting defects for eclipse,” in *Predictor Models in Software Engineering, PROMISE’07: ICSE Workshops, International Workshop on May, 2007.*

- [53] V. Basili, L. Briand and W.L. Melo, “A Validation of Object Oriented Design Metrics as Quality Indicators,” *IEEE Trans. Software Eng.*,1996.
- [54] Vasileios Apostolidis-Afentoulis, Konstantina-Ina Lioufi, “SVM Classification with Linear and RBF Kernels,” 2015.
- [55] W. Li and S. Henry, “Object-oriented metrics that predict maintainability,” *Journal of Systems and Software*, 23(2):111–122, 1993
- [56] W.Melo and F. B. E. Abreu, “Evaluating the impact of object oriented design on software quality,” in *Proceedings of the 3rd International Software Metrics Symposium*, pp. 90–99, Berlin, Germany, March, 1996.
- [57] Y. Singh, A. Kaur and R. Malhotra, R, “Empirical validation of object-oriented metrics for predicting fault proneness models,” *Software Quality Journal*, vol. 18, no. 1, pp. 3-35, 2010.
- [58] Zhongqiang Wei, Hongzhe Xu, Wen Li, Xiaolin Gui, and Xiaozhou Wu, “Improved Bayesian Network Structure Learning with Node Ordering via K2 Algorithm,” *Springer International Publishing Switzerland LNAI 8589*, pp. 44–55, 2014.

APPENDIX

APPENDIX 1: WEKA PREPROCESS SCREEN

The screenshot shows the Weka Explorer interface in the Preprocess tab. The 'Current relation' is 'allReliabilityFix' with 8093 instances and 22 attributes. The 'Selected attribute' is 'i_>software', which is a nominal attribute with 33 distinct values and 0 missing values. The 'Attributes' list on the left shows 16 attributes, with 'i_>software' selected. The 'Class' is set to 'bug (Num)'. A bar chart visualizes the distribution of the selected attribute, with the highest counts for 'camel' (965) and 'ArcPlatform' (234).

Current relation
Relation: allReliabilityFix
Instances: 8093
Attributes: 22
Sum of weights: 8093

Selected attribute
Name: i_>software
Missing: 0 (0%)
Distinct: 33
Type: Nominal
Unique: 0 (0%)

No.	Label	Count	Weight
1	log4j	205	205.0
2	lucene	340	340.0
3	synapse	256	256.0
4	xalan	909	909.0
5	ArcPlatform	234	234.0
6	Berek	43	43.0
7	camel	965	965.0
8	ckjm	10	10.0

Attributes
All None Invert Pattern

No.	Name
<input checked="" type="checkbox"/>	i_>software
<input type="checkbox"/>	wmc
<input type="checkbox"/>	dit
<input type="checkbox"/>	noc
<input type="checkbox"/>	cbo
<input type="checkbox"/>	rfc
<input type="checkbox"/>	lcom
<input type="checkbox"/>	ca
<input type="checkbox"/>	ce
<input type="checkbox"/>	npm
<input type="checkbox"/>	lcom3
<input type="checkbox"/>	loc
<input type="checkbox"/>	dam
<input type="checkbox"/>	moa
<input type="checkbox"/>	...

Status
OK Log x 0

APPENDIX 2: WEKA FILTER OPTIONS

Weka Explorer

Preprocess | Classify | Cluster | Associate | Select attributes | Visualize

Open file... | Open URL... | Open DB... | Generate... | Undo | Edit... | Save...

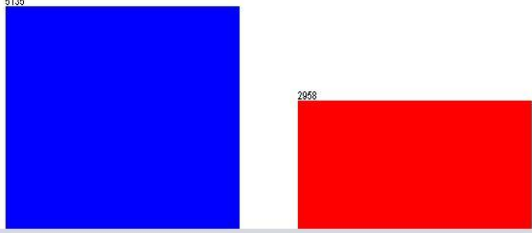
Filter: Choose **NumericToNominal -R last** Apply Stop

Current relation
Relation: allReliabilityFix2-weka.filters.unsupervised.attribute.NumericTo...
Instances: 8093
Attributes: 21
Sum of weights: 8093

Selected attribute
Name: bug
Missing: 0 (0%)
Distinct: 2
Type: Nominal
Unique: 0 (0%)

No.	Label	Count	Weight
1	0	5135	5135.0
2	1	2958	2958.0

Class: bug (Nom) Visualize All



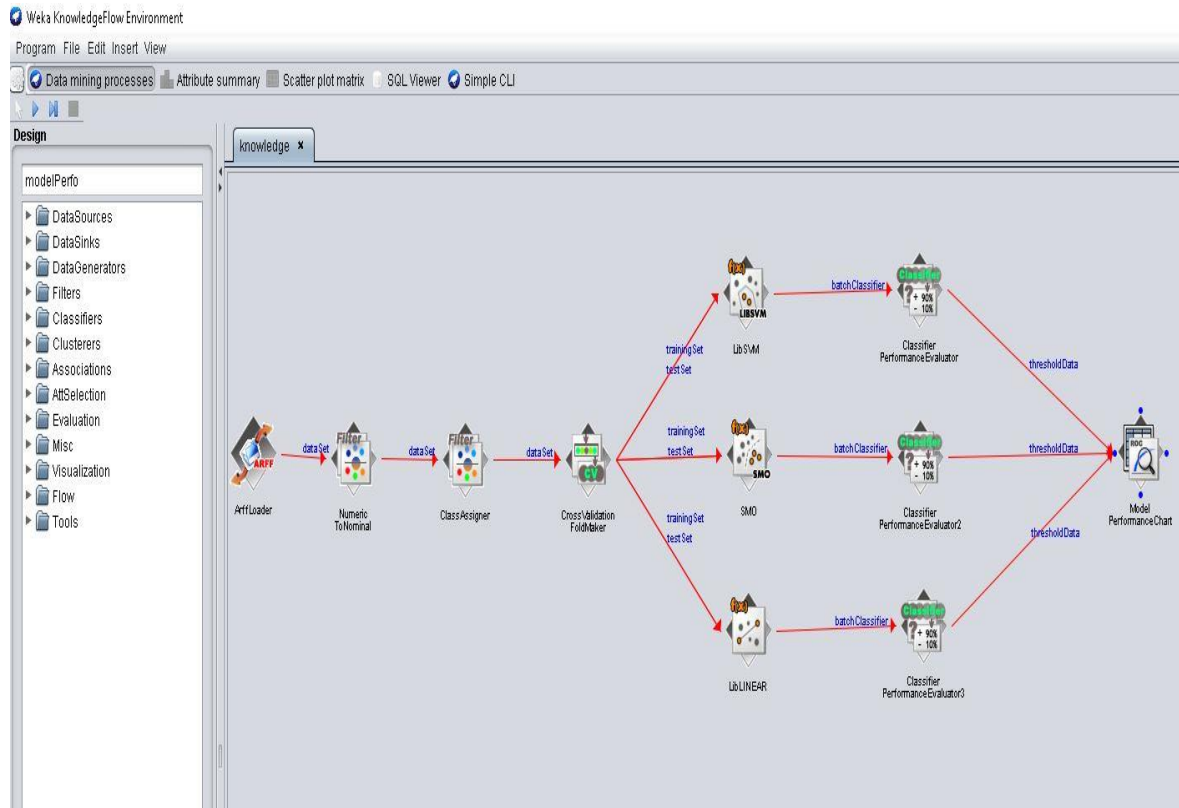
Attributes
All | None | Invert | Pattern

No.	Name
5	rfc
6	lcom
7	ca
8	ce
9	npm
10	lcom3
11	loc
12	dam
13	moa
14	mfa
15	cam
16	ic
17	cbm
18	amc
19	max_cc
20	avg_cc
21	bug

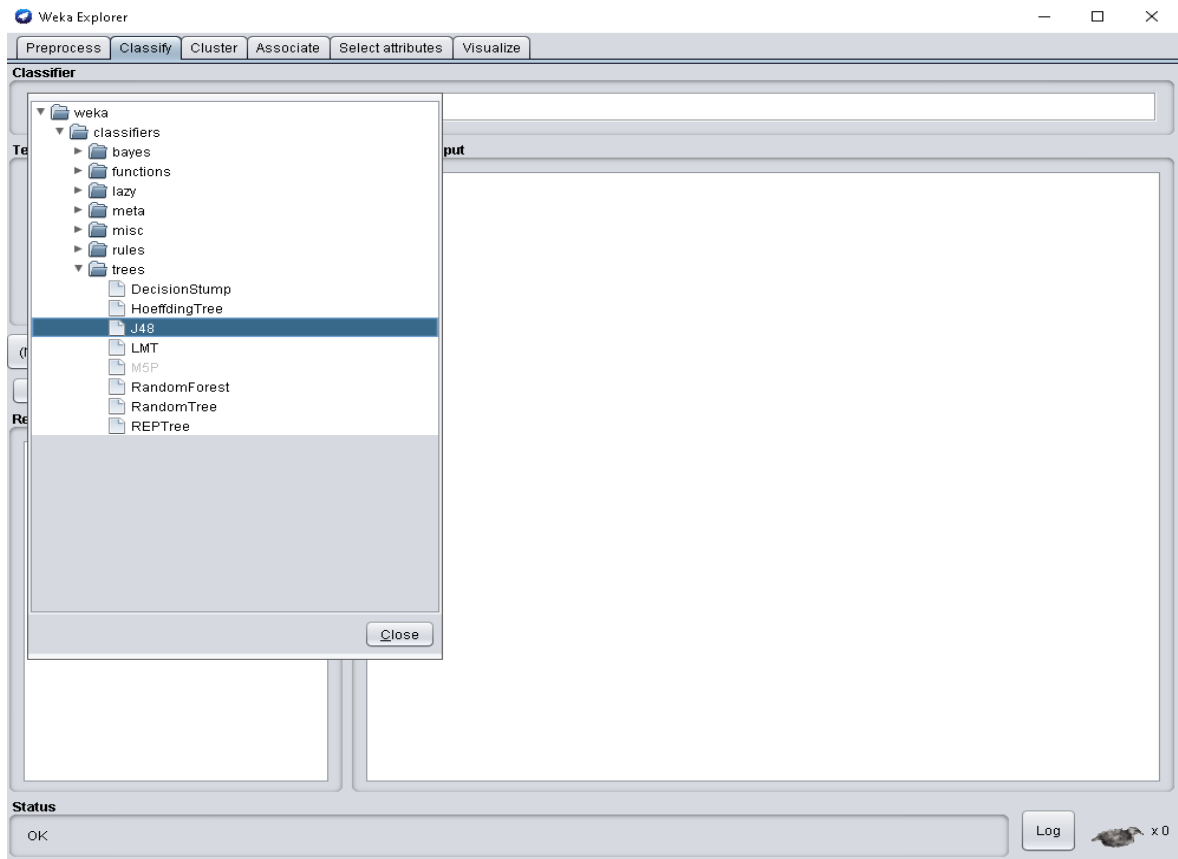
Remove

Status
OK Log x 0

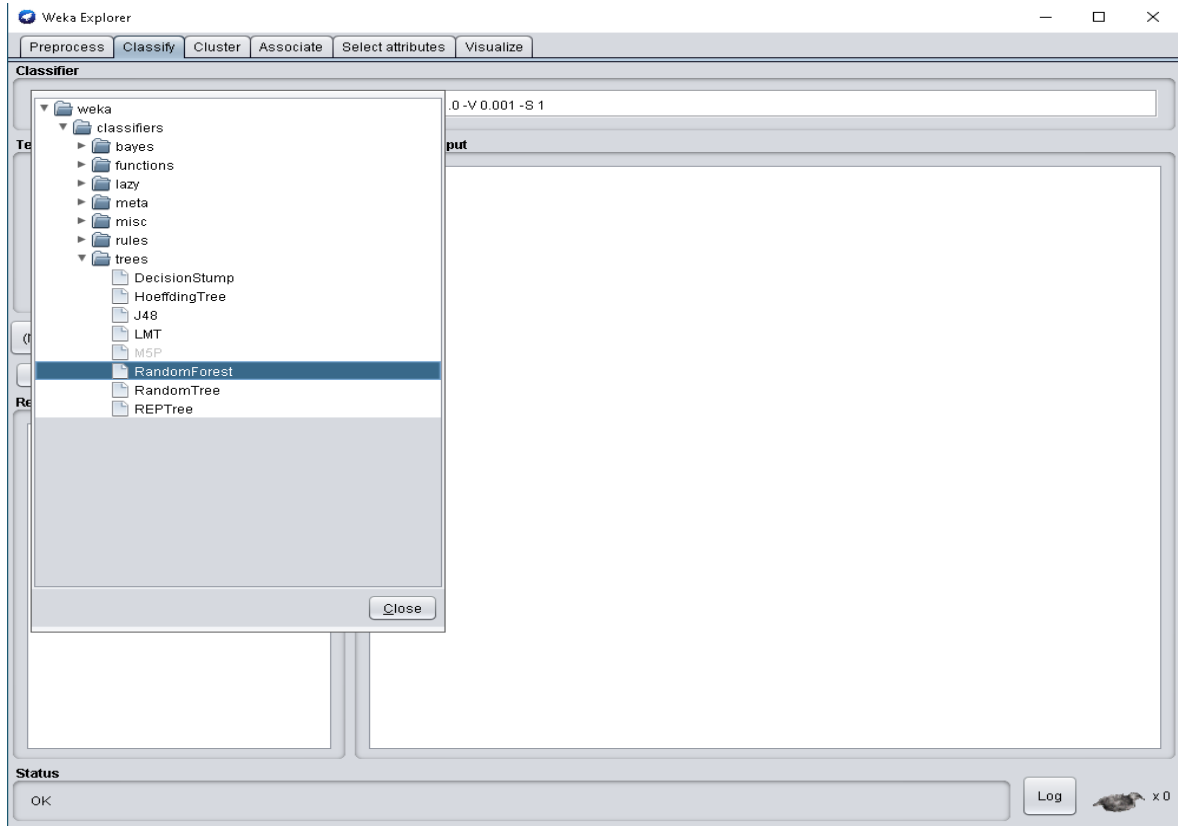
APPENDIX 3: WEKA KNOWLEDGE FLOW



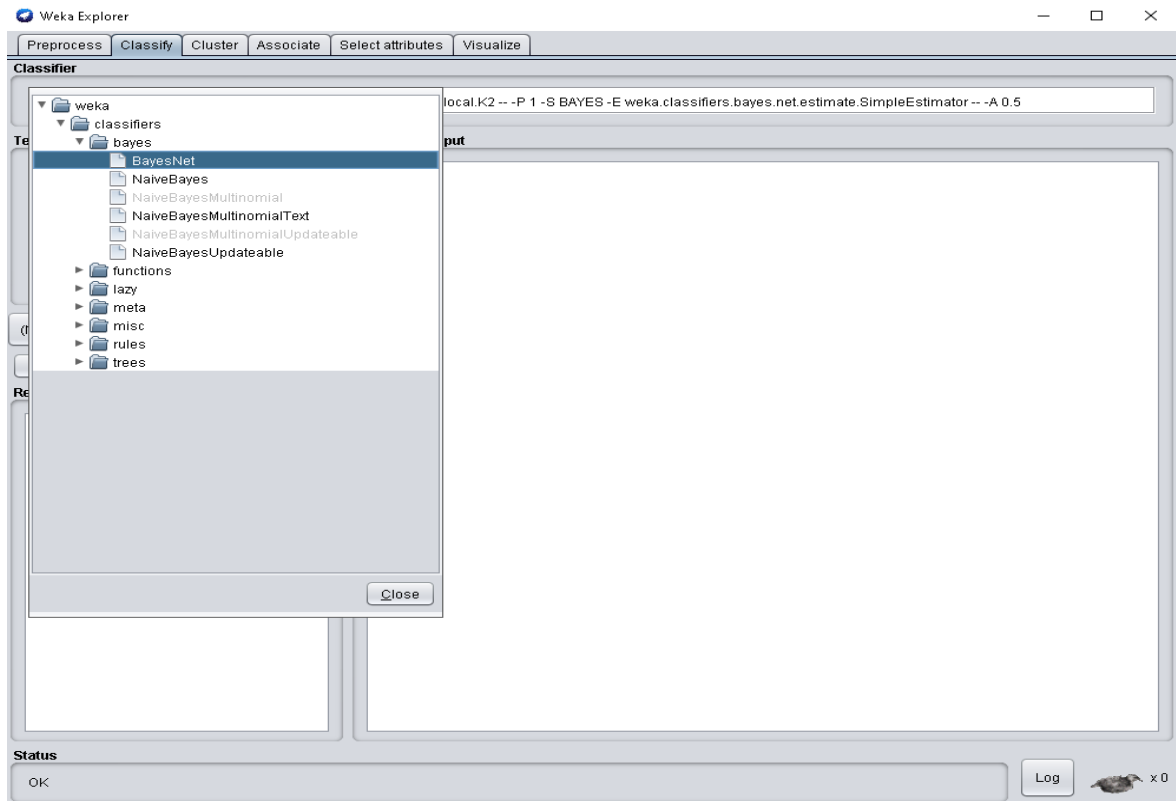
APPENDIX 4: DECISION TREE



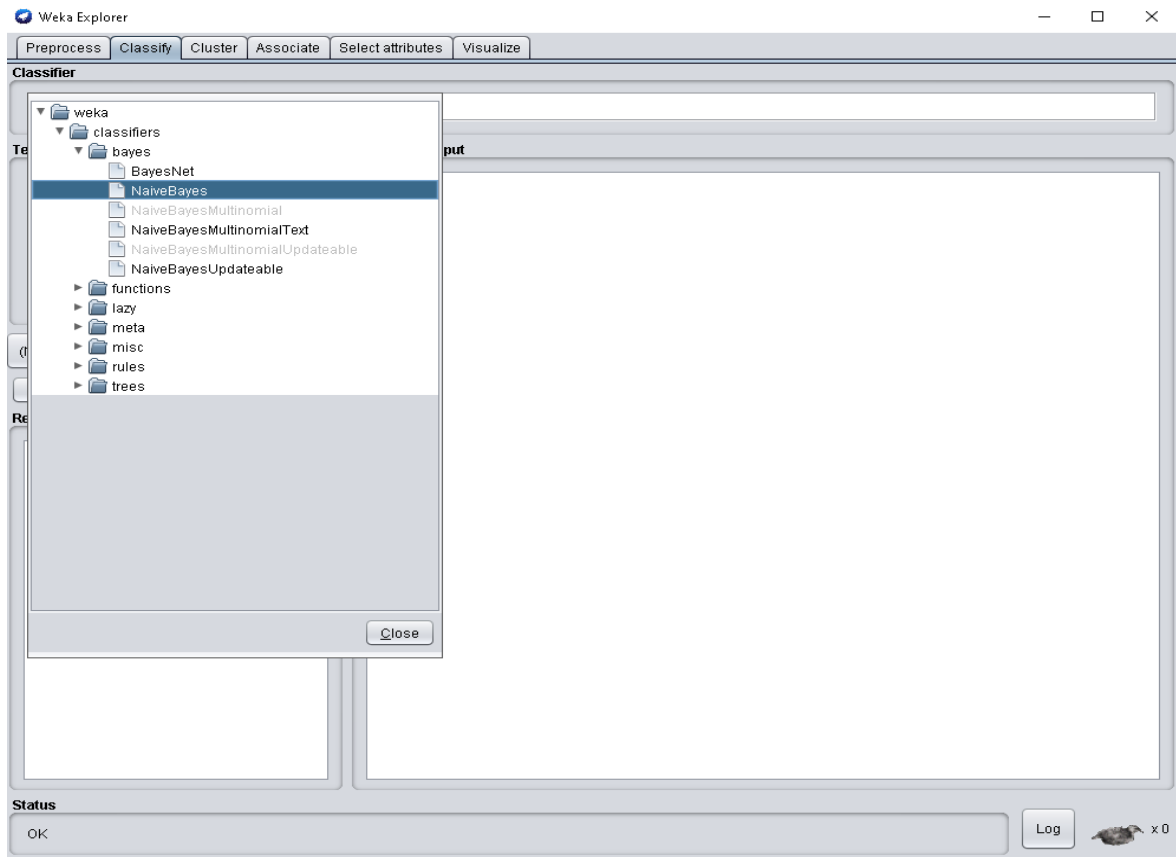
APPENDIX 5: RANDOM FOREST



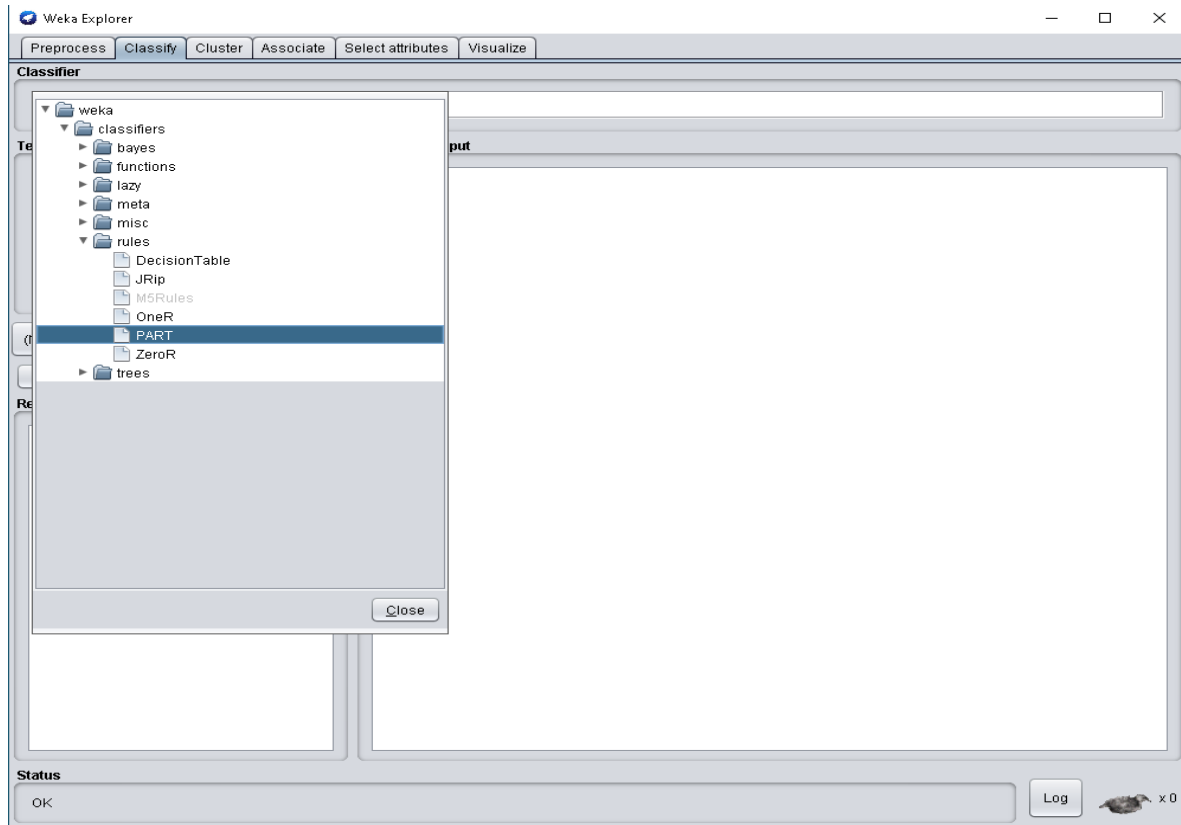
APPENDIX 6: BAYESIAN NETWORK



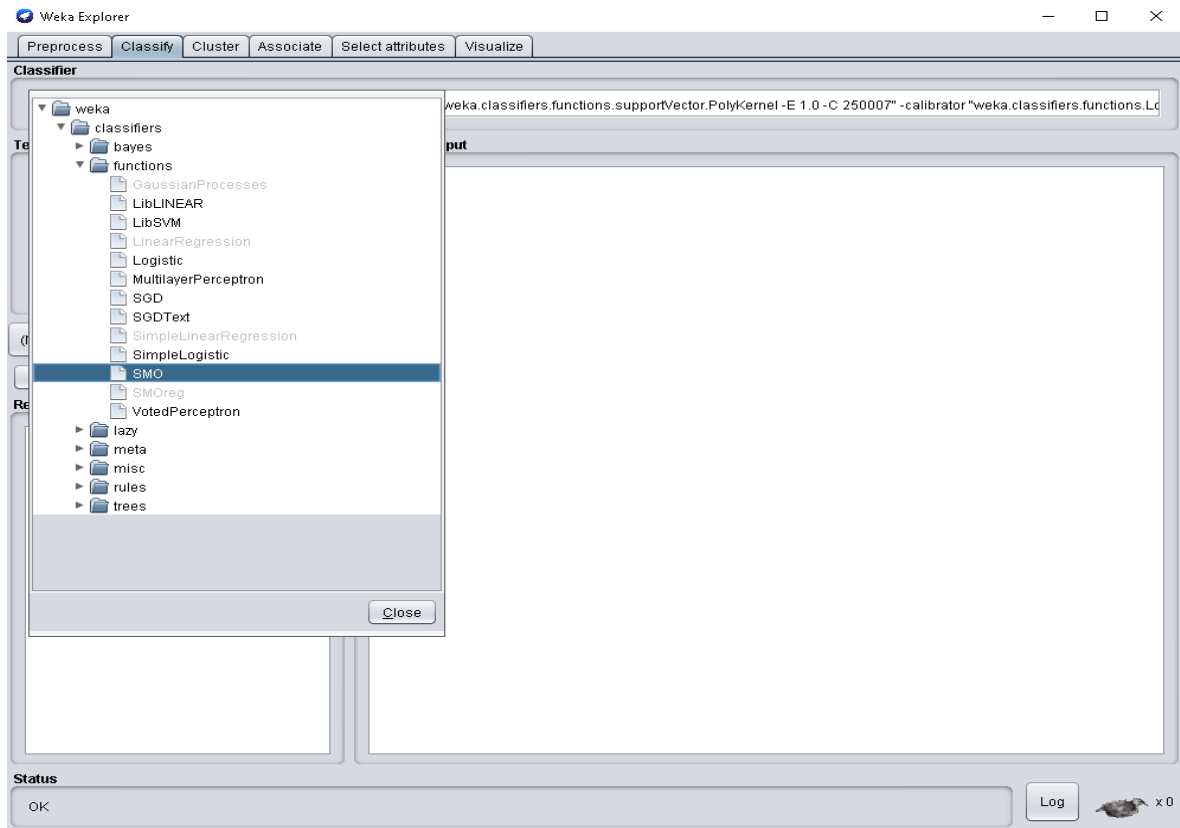
APPENDIX 7: NAÏVE BAYES



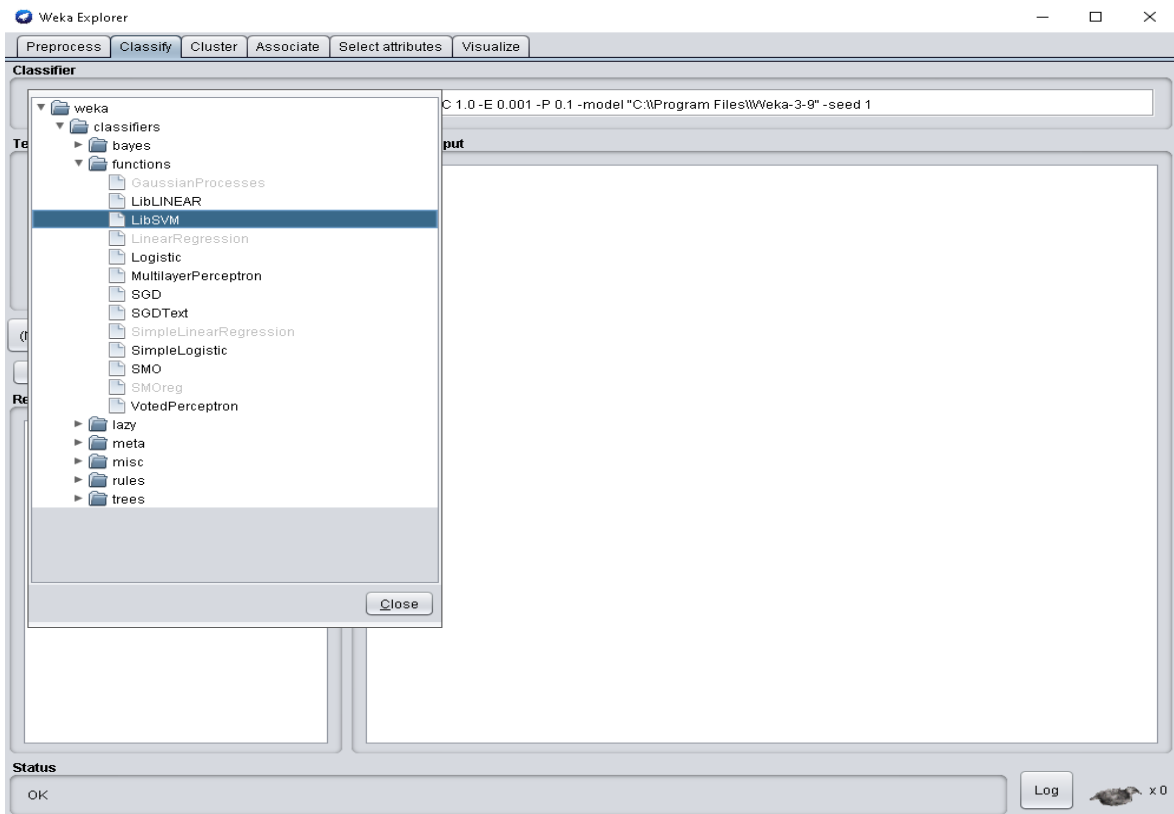
APPENDIX 8: RULE BASED CLASSIFICATION



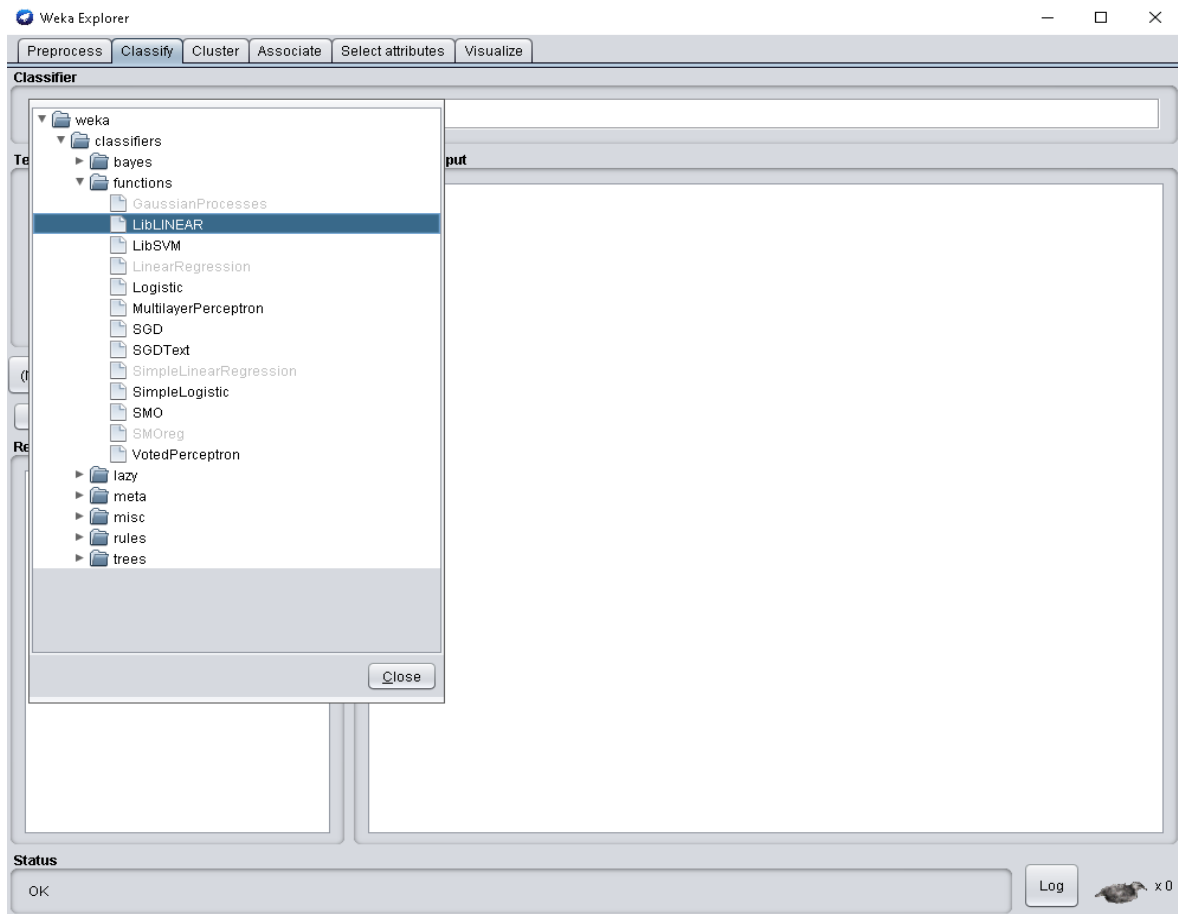
APPENDIX 9: SMO CLASSIFICATION



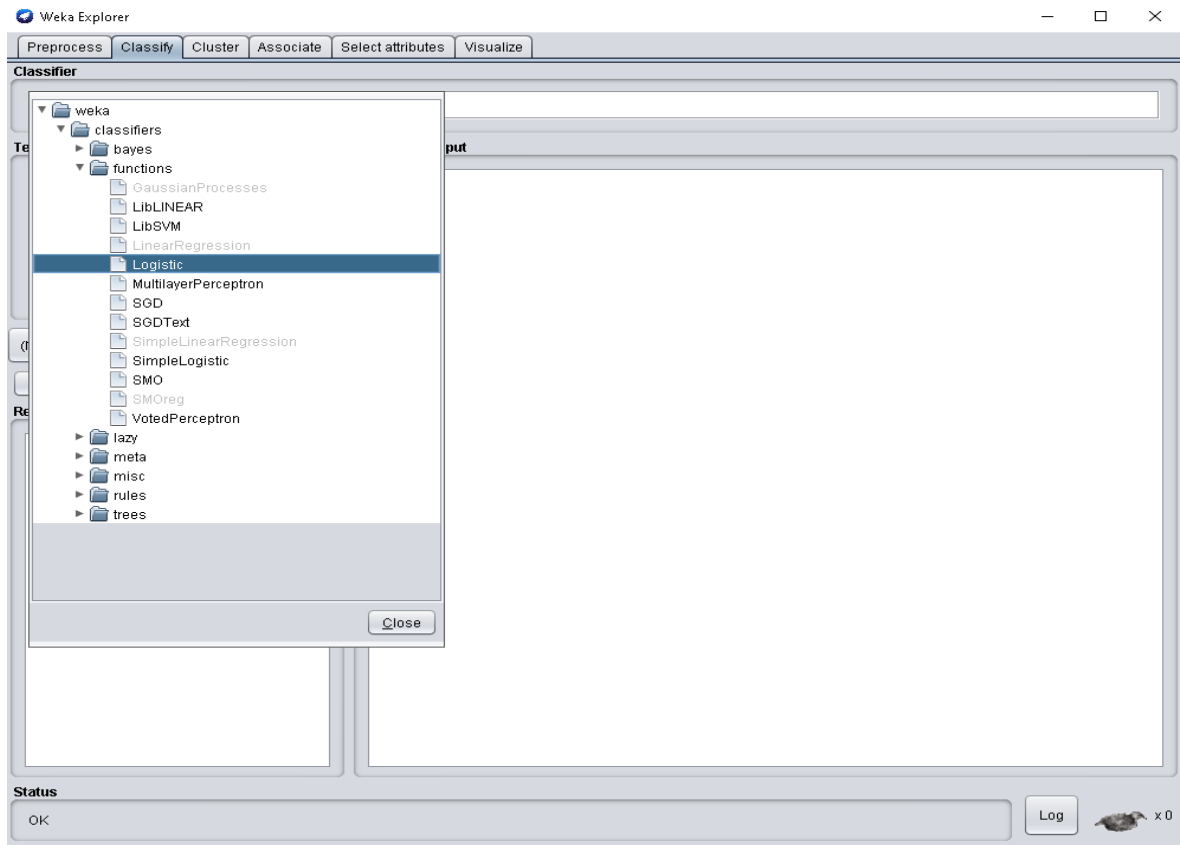
APPENDIX 10: LIBSVM



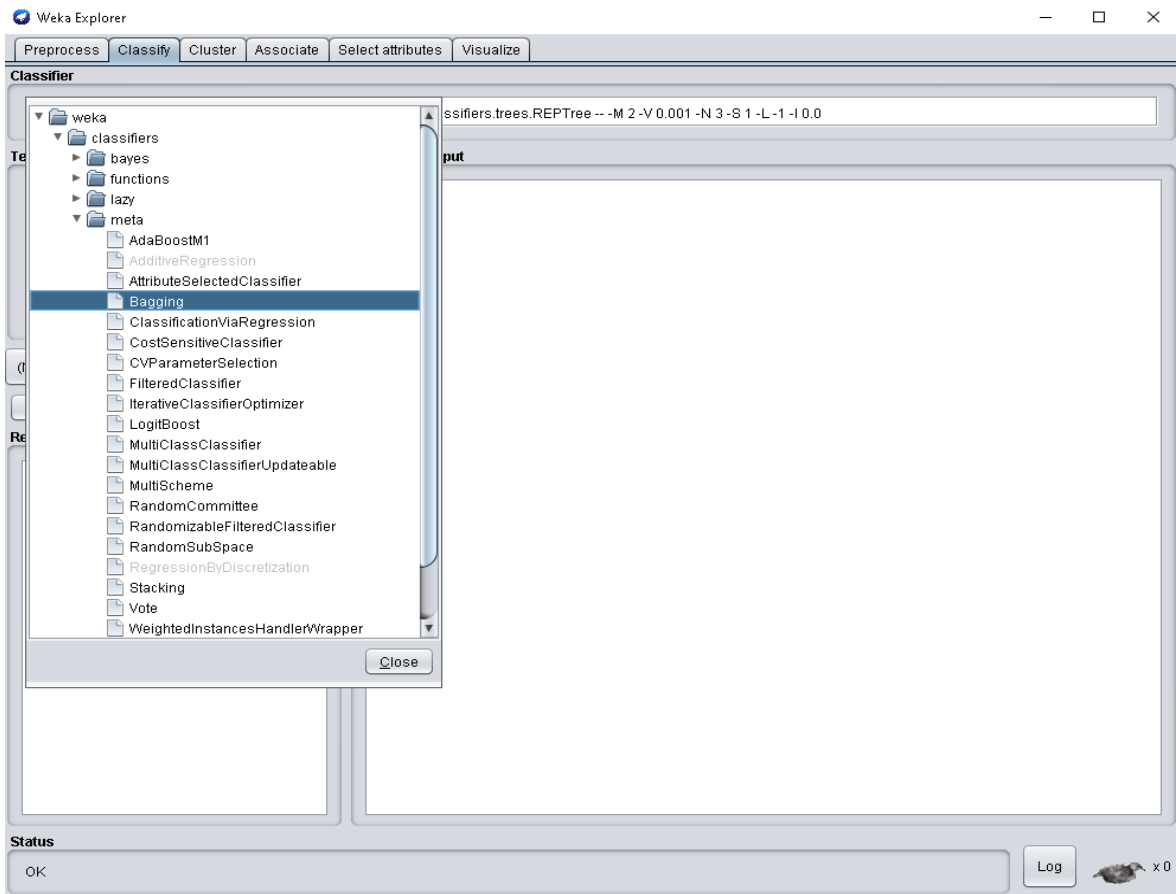
APPENDIX 11: LIBLINEAR



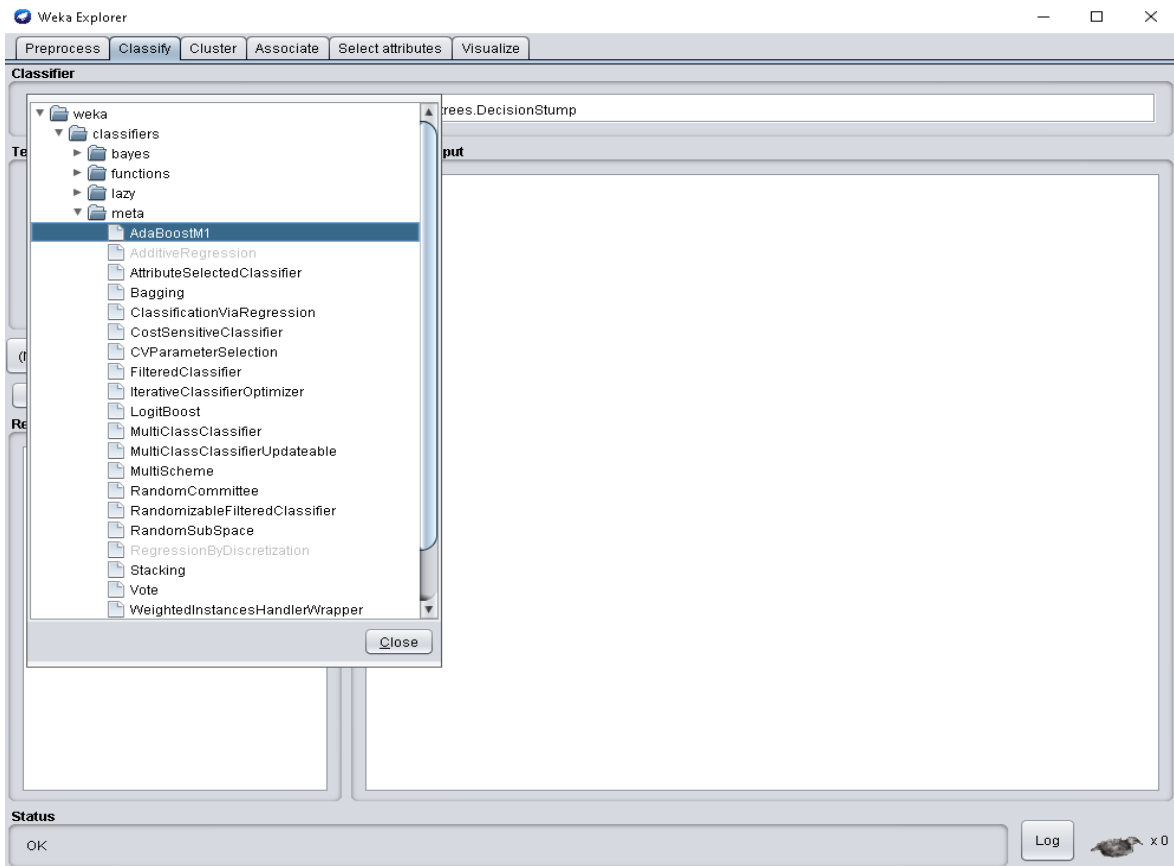
APPENDIX 12: LOGISTIC REGRESSION



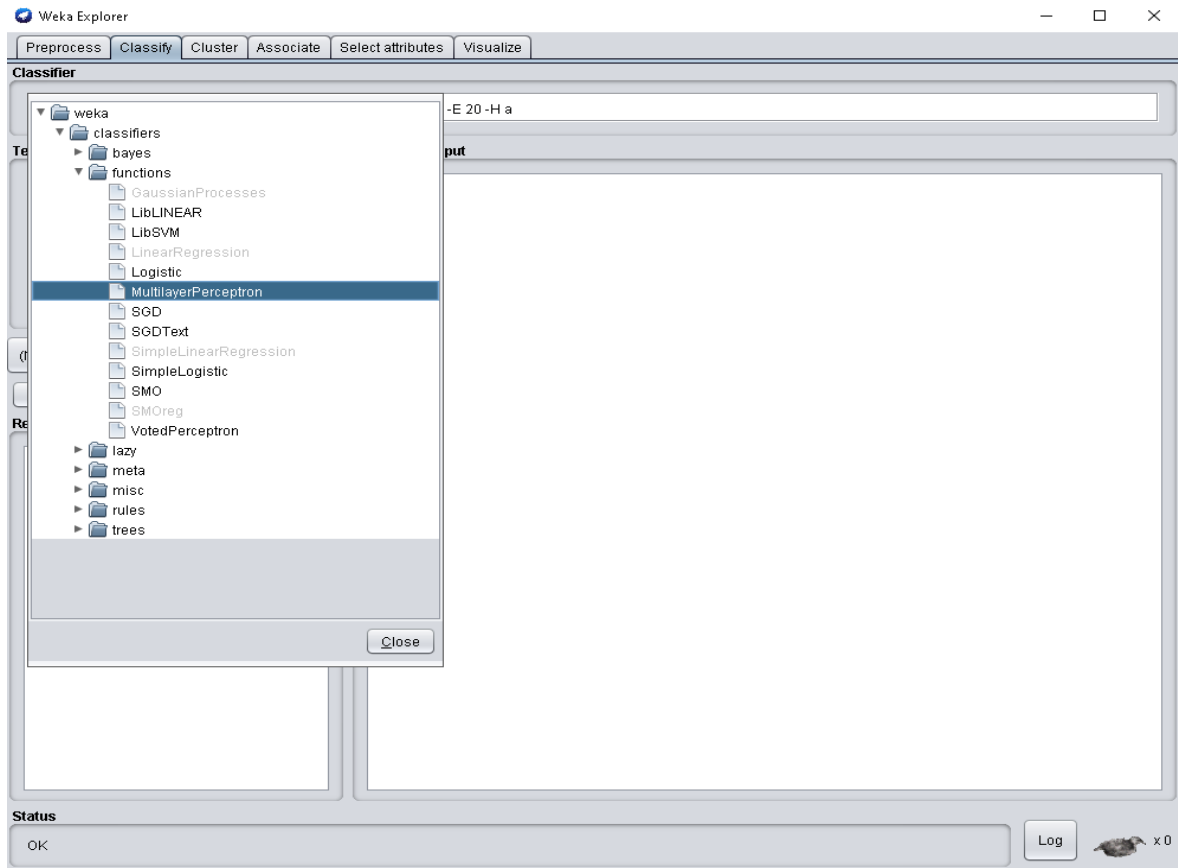
APPENDIX 13: BAGGING



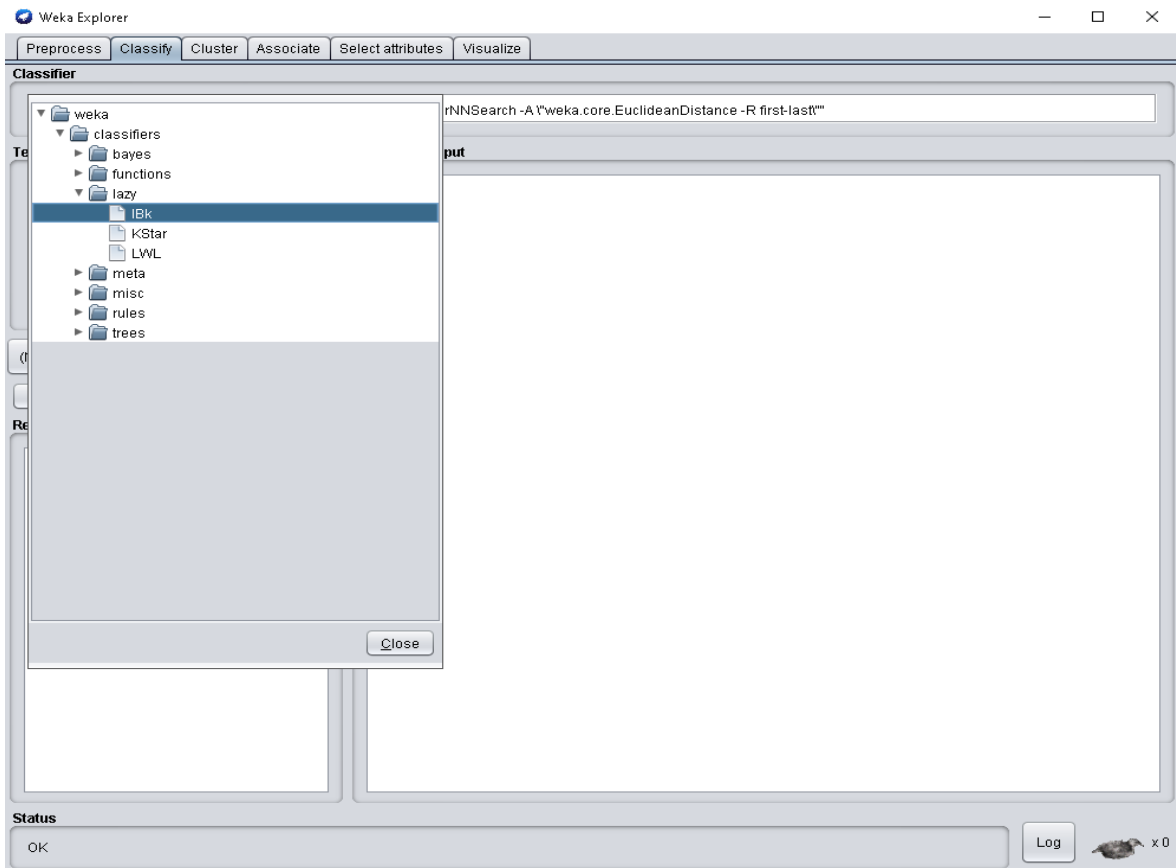
APPENDIX 14: BOOSTING



APPENDIX 15: ARTIFICIAL NEURAL NETWORKS



APPENDIX 16: NEAREST NEIGHBORS



APPENDIX 17 RAPIDMINER

File Edit Process View Connections Settings Extensions Help

Views: Design Results Turbo Prep Auto Model Deployments

Repository

Import Data

- Training Resources (connected)
 - Samples
 - Community Samples (connects)
 - DB (Legacy)
 - Local Repository (ozzan.ilhan)
 - Temporary Repository (ozzan.ilhan)

Operators

Search for Operators

- Data Access (53)
- Blending (81)
- Cleansing (29)
- Modeling (160)
- Scoring (14)
- Validation (30)
- Utility (87)
- Extensions (133)

[Get more operators from the Marketplace](#)

Process

Process

Welcome to RapidMiner Studio!

Start Recent Learn

Start a new project

- Blank**
Start a new process from scratch in the design view.
- Turbo Prep**
Prepare your data interactively; transform, clean and combine data sets.
- Auto Model**
Build and optimize models using automated machine learning.

Choose a template to start from

- Churn Modeling**
Predict which of your customers will churn and why with a decision tree.
- Direct Marketing**
Predict response to campaigns and increase the conversion rate of your campaign.
- Credit Risk Modeling**
Model credit default risk by training an optimized Support Vector Machine (SVM) model.
- Market Basket Analysis**
Find products frequently purchased together and turn them into rules for recommendations.
- Predictive Maintenance**
Model equipment failures to schedule maintenance pre-emptively.
- Price Risk Clustering**
Cluster price developments using X-Means to unveil price-risk-relationships.
- Lift Chart**
Create a lift chart to visualize the improvement that a model provides compared to guessing.
- Operationalization**
Embed predictive models into business processes to trigger the right actions automatically.
- Outlier Detection**
Detect anomalies in data resulting from a chemical analysis of wines.
- Geographic Distances**
- Medical Fraud Detection**
- Web Analytics**

Recommended Operators

- Retrieve (12%)
- Select Attributes (6%)
- Set Role (5%)
- Apply Model

