Manual Testing Execution Time Estimation

By

Muhammad Imran Khawar

Fall 2015-MS-15-(CSE)

00000119432

Supervisor

Dr. Wasi Haider Butt

DEPARTMENT OF COMPUTER ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

July, 2019

Manual Testing Execution Time Effort Estimation

By

Muhammad Imran Khawar

Fall 2015-MS-15-(CSE)

00000119432

A thesis submitted in partial fulfillment of the requirements for the degree of

MS Computer Software Engineering

Thesis Supervisor:

Dr. Wasi Haider Butt

Thesis Supervisor's Signature: ____________________________________

DEPARTMENT OF COMPUTER ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,

ISLAMABAD

July, 2019

Declaration

I hereby certify that I have developed this thesis titled as “*Manual Testing Execution Time Effort Estimation*” entirely on the basis of my personal efforts under the sincere guidance of my supervisor Dr. Wasi Haider Butt. All of the sources used in this thesis have been cited and contents of this thesis have not been plagiarized. No portion of the work presented in this thesis has been submitted in support of any application for any other degree of qualification to this or any other university or institute of learning.

Signature of Student
Muhammad Imran Khawar
Fall 2015-MS-15-(CSE)
00000119432

Language Correctness Certificate

This thesis has been read by an English expert and is free of typing, syntax, semantic, grammatical and spelling mistakes. Thesis is also according to the format given by the university.

Signature of Student

Muhammad Imran Khawar

Fall 2015-MS-15-(CSE)

00000119432

Signature of Supervisor

Dr. Wasi Haider Butt

Copyright Statement

Acknowledgements

I am thankful to my Creator Allah Subhana-Watala to have guided me throughout this work at every step and for every new thought which You setup in my mind to improve it. Indeed I could have done nothing without Your priceless help and guidance. Who so ever helped me throughout the course of my thesis, whether my parents or any other individual was Your will, so indeed none be worthy of praise but You.

I am profusely thankful to my beloved parents who raised me when I was not capable of walking and continued to support me throughout in every department of my life.

I would also like to express special thanks to my supervisor Dr. Wasi Haider Butt for his help throughout my thesis and also for Requirement Engineering course which he has taught me. I can safely say that I haven't learned any other engineering subject in such depth than the ones which he has taught.

I would also like to express my gratitude to my very kind Engr. Dr. Shoab A Khan, Head of Department of Computer Engineering; from the deep core of my heart for all he has done for me to complete this work. Without his kind advice, encouragement, guidance and support, it was impossible for me to carry out this task.

I would also like to thank Dr. Urooj Fatima, Dr. Arsalan Shaukat and Dr. Usman Akram for being on my thesis guidance and evaluation committee.

Finally, I would like to express my gratitude to all the individuals who have rendered valuable assistance to my study.

Abstract

In software development practices of software industry. Testing plays a crucial role in the software development life cycle and it does so by verifying and validating the software’s quality.

Since testing is thought to be a high-cost process in software development and the fact that the budget for testing the product is limited, finding the execution time needed for software testing activities like the prioritization, scheduling and progress monitoring of test cases is of significant importance.

Manual testing remains a prevailing and significant approach to validating software applications, particularly in certain areas such as domain-specific testing. To carry out test planning, prioritization and progress tracking, it is essential to know the execution time of test instances. In this work, we apply, assess and present on the basis of the specifications of test for execution time estimation and prediction of manual test-cases. Our method operates by extracting timing data in manual test specification for different steps. This data is then utilized for prediction of maximum time for those test steps that were not executed. An approach for time of execution prediction and estimation for manual test-cases is suggested in this thesis. The technique utilizes test specifications and historical information accessible from test instances that were previously performed. Our strategy works by acquiring timing data from each and every step of test instances earlier carried out.

The gathered data is used from their test specifications to predict the execution period for non-executed test instances. Classification test case scores are extracted from the test specification contained in the Test Manager Tool and plotted with the acquired timing information. Classification is used on Test Cases to estimate the execution period of non-executed sample instances after estimating the time from this mapping. LMKR performed a case survey where the suggested technique is applied and the outcomes are validated. The results obtained show that the predicted time of execution of studied test cases is close to their actual time of execution.

Key Words: *Test Case Point (TP); Quality Assurance (QA); Function Point Analysis (FPA); Software Test Estimation; Test Case Management; Source Lines of Code (SLOC)*

Table of Contents

List of Figures

List of Tables

CHAPTER 1: INTRODUCTION

1.0 Introduction

Software testing is significant for delivering high quality software projects [1, 2]. Program testing is overall costly and a time taking procedure. It is even truer for manual testing which requires a tester to complete tests on the system under test (SUT) without utilizing automation [3]. Test automation is currently turning into a well-known technique to cut the expenses related with software testing; a computer can pursue and finish the testing quicker than a human being and can finish the tests medium-term to introduce the outcomes in the first part of the day [4, 5]. The time and exertion spared in actual testing can be spent on composing the program for testing. Automatic testing may require more effort first and foremost relying upon the sort of utilization to be tried and the automation tools that are picked [6]. Furthermore, test automation right now can't contend with human instinct, derivation, inductive thinking, nor would it be able to change ways in a test to examine something that had not been recently estimated [1]. In this way, manual testing still has a significant job in the product testing procedure and it is hard to supplant it totally.

In the course of recent years, the industry has begun to take interest for finding approaches to improve efficiency and quality of testing [7, 8]. Effective software testing procedures cut the general expenses of testing and result in prior issue recognition [9, 10, 11], and furthermore consider a few factors, for example, inclusion, test technique, test execution, arranging and investigation of test outcomes. [12, 13]. In this point of view, test choice and prioritization are connected to one another and are quickly turning into an indistinguishable piece of a general testing strategy [14]. So as to do test choice, prioritizing and scheduling, there are a few proposed strategies [15, 11, 9, 16], where the underlying issue of test prioritization is recognized as a multi criteria problem, which means a specific criteria must be created before execution. The execution time is a key factor that can effect test booking, prioritization and execution observing

In our previous works [17, 18], we suggested a multi criteria decision support system (DSS) for selection and prioritization of manual test cases for integration while taking other factors such as time, dependency, coverage into consideration.

This research paper is an enhanced version of [19], which presents a new methodology for evaluating and predicting the implementation time of manual tests depending upon specifications and historical data available on the past tests. The execution time is organized into following two categories: 1) Maximum Time (MT) and 2) the Automatic Time (AT) whereas MT stays constant while AT can vary. Afterwards we have given formal definitions of MT and AT. Our suggested technique separates timing data for different activities. It is an integral step of our technique that language parsing of specifications is done to recognized word groupings which are then used accordingly for checking if current timing data on various test exercises is accessible or not. In addition, regression models are employed greatly to anticipate the actual time for manual tests.
Our suggested technique for surveying and anticipating the execution time of software testing is as follows: We quantified the time of manual tests through perceiving few essential components in tests, examining the historical test data. Since the actual time for tests is on a very basic level a time dependent on system, the Polynomial and spline regression representations are utilized for giving an estimate of this time. Likewise, the forecast blunder of the relapse models has been evaluated in order to survey the expectation calculations. A study at LMKR is moreover done to gauge the suggested technique.[20]. The association of this postulation is spread out as pursues: Section 2 gives an establishment of the underlying issue and besides an audit of research on execution time figure, Section 3 depicts the proposed strategy. An advanced relevant examination has been arranged in Section 4, threats to validity and limitations are discussed in Section 5. Section 6 is about the discussion and some future course of the present work and finally Section 7 wraps up the whole thesis.

This chapter covers the motivation for the selection of topic and background works and problem statement.

Background

Testing is one of the most important activity in software development [7] for both the developers and the software users. It is a process in which a software product is being analyzed to compare and distinguish the actual and expected results. In other words, it is a process of identifying faults in the software. It is also used to check the quality of software product. Software testing ensures the reliability of the product. In order to ensure the maximum reliability of working software, researchers have identified several techniques at different testing levels namely unit, integration and system testing. A software is composed of several modules and such modules are separately tested in Unit testing. It requires deep knowledge about the modules and it can be performed in the initial stages of the development of software. Only 65 % of errors can be detected in this level of testing [8]. The next level of testing is the integration testing. It is also known as integration and test (I&T). In this testing level, different modules are integrated and tested for errors. It aims at testing the variable exchange between the modules and inter-module communication [9]. In System testing, complete working of the software is tested for the user-specified requirements. Also, it is tested whether the software is acceptable in the market, which is also termed as Acceptance testing. This research work will focus on the test cases from integration testing. In the process of Software Development Life Cycle (SDLC), testing of software is an important phase [10]. It is considered as an expensive and time consuming activity which requires more than 50% of software development cost. This percentage will be higher in safety critical applications. A software is composed of several units called as modules or components. After the individual units are tested separately, it is necessary to combine them and see how they interact with each other to form a complete system. If system still has defects, then the severity of the defect decide whether to proceed onto the integration phase or to run both the unit testing and integration testing in parallel. For example, there may exist some error like "404: Page not found", which makes the system incomplete and leads to integration issues. During integration testing, hardware integration with the software is also tested. This is to test how the hardware behaves on the software under test. The test plan for the integration testing will be written before performing integration testing. One of the part of integration testing is interface testing which is used to test the interface between components or modules in a system [11]. A test specification consists of number of test cases. In the manual testing there are number of steps in the single test case. The test steps consists of actions, input data and expected output from the system. These actions are the activities conducted in the system being tested. This is to ensure that it has same behavior of the system for expected and in the actual.

1.1 Problem Statement behavior

The aim of this research is to perform estimate manual testing execution time estimation. In order to achieve first we need to collect the test case data sets then perform the Test Point Analysis

Given a program Prog = {P1, P2, … , Pn} contains of a set of namespace N, a namespace that contains of a set of classes N = {C1, C2, … , Cm}, a class contains of set of methods C = {M1, M2, … , Mk}.

A set of test-cases is described to check the program for bugs including class, techniques, number and kinds of inputs, excepted outputs, and status. The test case set is described as T = {t1, t2, … , tx}, where ti = { a1, a2, a3, …., at } and type (a) {Integer, string, double, bolean, char, byte} is a test-case described to cover one of the following:

Class Coverage: $Ccov(t_i) = \{C_1, C_2, \dots, C_{m1}\} \subseteq C$ covered by t_i

Method Coverage: $Mcov(t_i) = \{M_1, M_2, \dots, M_{k1}\} \subseteq M$ covered by t_i

Input Coverage: Icov(ti) = |a| covered by ti

In order to identify the number of generated complex test cases, two classifiers are used to find out whether a test-case is complex or normal. The two methodologies are being used Naïve ayesian and the other one is Decision Tree. For prediction of status of the test-cases we used these three principles. Specifically, a test-case is marked as complex if it is satisfied with all of the three principles, else; the test-case is declared to be normal. The three principles are:

$\forall\ (i,j)$ where $i \neq j$, $(Ccov(t_i) = Ccov(t_j)) \subseteq C$

$\forall\ (i,j)$ where $i \neq j$, $(Mcov(t_i) = Mcov(t_j)) \subseteq M$

$\forall\ (i,j)$ where $i \neq j$, $Icov(t_i) = Icov(t_j))$

1.2 Structure of Thesis Report

The structure of this thesis report is as follows:

- Chapter 2 presents the methodology used for the thesis. It also focus on the literature study in the area of software effort estimation and drawbacks of the methods identified from the literature. Another methodology used for this thesis is case study. An overview and design of the case study is also defined in this section.
- Chapter 3 describes the architecture of the implemented system to predict and estimate the execution time of test cases. It focuses on the creation of a database and the source from which the data are extracted to build our database. Implementation of the proposed algorithm is also explained in this section.

- Chapter 4 presents the input test cases used to test our implemented algorithm and the obtained result. The results are evaluated and presented in this section.
- Chapter 5 discusses the challenges faced during the implementation, limitations of the proposed approach and some of the techniques used to overcome the limitations.
- Chapter 6 concludes the report with some future works.

CHAPTER 2: LITERATURE REVIEW AND RESEARCH METHODOLOGY

The first phase in this thesis work is the literature review of existing studies conducted in the area of test effort estimation followed by an industrial case study.

2.1 Literature Review

One of the research techniques we used in this thesis report was the systematic literature review, to identify the related work and existing approaches used for time of execution prediction and estimation of test-cases. This review helped in finding a suitable method for prediction and estimation of the time execution from manual test-cases written English which is a natural language.

The databases used for the literature review are as following:

• Scopus

• IEEE Xplore

• Google Scholar

In this process, we started with Scopus followed by IEEE Xplore and Google Scholar to find relevant literature. The selection of these sources is motivated by its extensive coverage of different types of academic sources such as scientific journals, books, conference papers and so on. The queries used for searching are as following:

- (((Prediction) OR (Estimation)) AND (Time) AND (Test))
- (((Prediction) OR (Estimation)) AND ((Time) OR (Effort)) AND (Test))
- (((Prediction) OR (Estimation) OR (Guess)) AND (Time) AND (Test))
- (((Prediction) OR (Estimation)) AND (Time) AND ((Test) OR (Execution)))
- (((Prediction) OR (Estimation)) AND (Time) AND ((Test) OR (Execution) OR (Execute)))
- (((Prediction) OR (Estimation)) AND ((Time) OR (Effort)) AND ((Test) OR (Cases)))
- (((Prediction) OR (Estimation)) AND (Time) AND (Test) AND (Manual))

These search queries have resulted in 67,684 papers. To narrow down the search, the papers which were published in “Computer Science” subject area and during the last five years have been selected. This reduced the sample size to 1,282 papers. From the obtained results, each and every papers was analyzed based on the title. If the title is related to our search, then in order to obtain more information, the abstract, introduction and conclusion of the documents were carefully read. By the process, relevant papers in the related field of test effort prediction or estimation were identified. When a paper in the field of our study is obtained from Scopus, the citations to that paper have been found using Google Scholar. The papers relevant to our search is obtained using “cited by” feature of Google Scholar. 96 relevant papers were gathered and analyzed. Among them, 12 were identified for our study as they were in the field of test effort prediction and estimation. The overall process followed for the literature review is depicted in Figure 1.

2.1.1 Related Work

In [16], Nageswaran et al., proposed a technique called Use Case Point (UCP). In this method, use cases are considered for the estimation of time for execution of test. Initially, the number of actors and use cases are identified for this approach. In addition to that, requirements of software, factors used either technical or environmental for estimation and time for execution of time. Using this technique, the estimation of the test effort can be measured in the initial level of development of software because it does not depend on LOC. This UCP strategy is utilized for estimation of exertion for testing effort all in all which incorporates test-plan, structure, execution, observing and revealing. In any case, this technique isn't utilized to catch every single occurrence of test action, for example, single test-case execution [17].

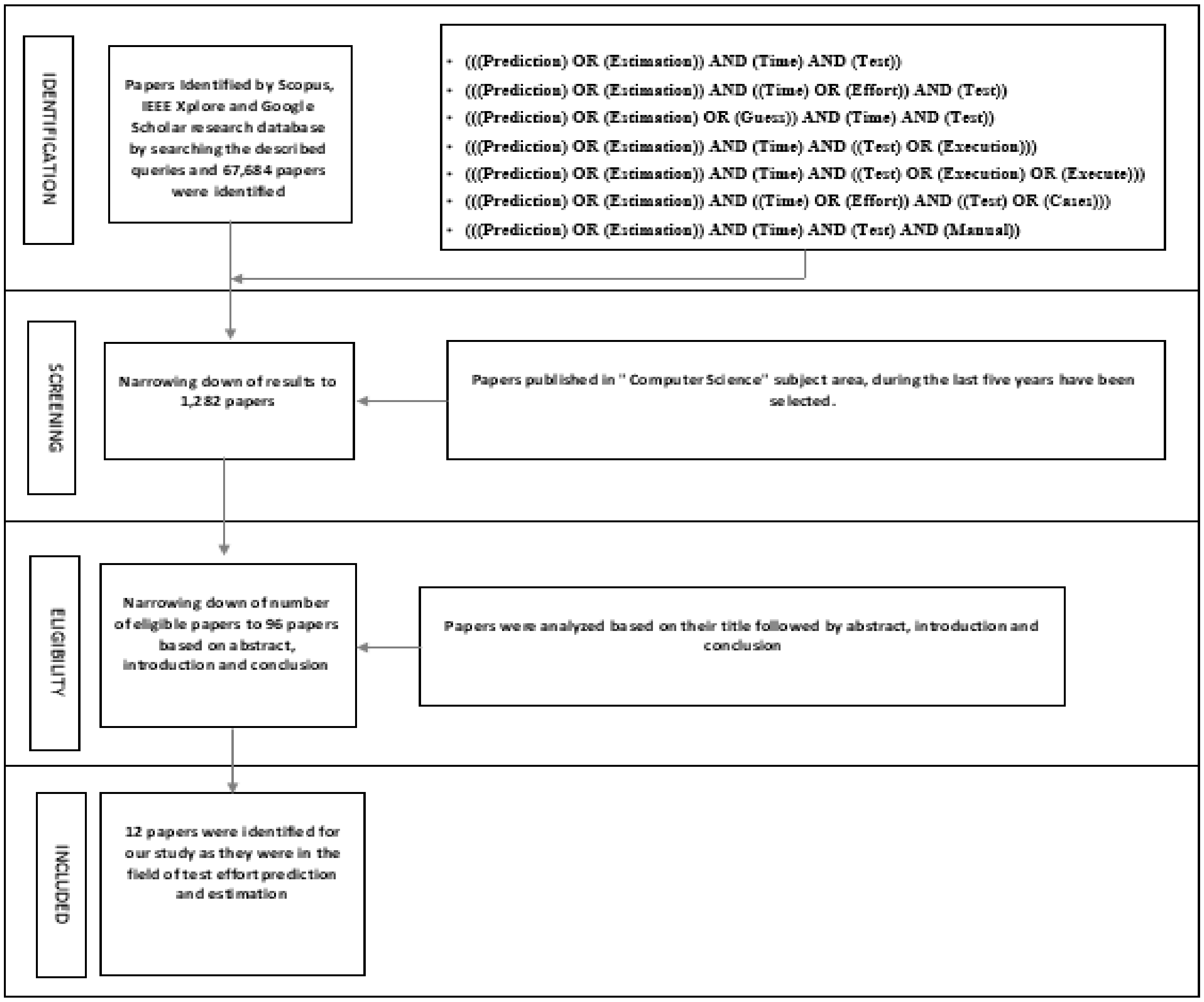

IDENTIFICATION
Papers Identified by Scopus, IEEE Xplore and Google Scholar research database by searching the described queries and 67,684 papers were identified
(((Prediction) OR (Estimation)) AND (Time) AND (Test))
(((Prediction) OR (Estimation)) AND ((Time) OR (Effort)) AND (Test))
(((Prediction) OR (Estimation) OR (Guess)) AND (Time) AND (Test))
(((Prediction) OR (Estimation)) AND (Time) AND ((Test) OR (Execution)))
(((Prediction) OR (Estimation)) AND (Time) AND ((Test) OR (Execution) OR (Execute)))
(((Prediction) OR (Estimation)) AND ((Time) OR (Effort)) AND ((Test) OR (Cases)))
(((Prediction) OR (Estimation)) AND (Time) AND (Test) AND (Manual))
SCREENING
Narrowing down of results to 1,282 papers
Papers published in "Computer Science" subject area, during the last five years have been selected.
ELIGIBILITY
Narrowing down of number of eligible papers to 96 papers based on abstract, introduction and conclusion
Papers were analyzed based on their title followed by abstract, introduction and conclusion
INCLUDED
12 papers were identified for our study as they were in the field of test effort prediction and estimation

Figure 2.1 Flow chart showing the process followed for literature review

In [18], an approach similar to UCP [16] has been proposed that is known as cuckoo technique for searching. This approach is also applied on use-cases for effort estimation of testing. This mainly differentiate search technique cuckoo and UCP. The UCP assigns weightage or a fixed value to their parameters such as actors or use-cases, whereas cuckoo search assigns a range of values for each and every parameters. This range of values assigned on the parameters can be either static (fixed) on dynamic (changing). It depends on the target system in which the estimation is done. This methodology additionally assess the ability of the team of development and testing. Based on all these parameters, test effort is estimated. So as to apply this strategy for assessing the effort of another undertaking, we need chronicled information from at any rate one anticipate.

Since the information is required from both the designers and analyzers, it depends on human for effort estimation.

In order to use this approach for estimating the effort of a new project, we require previous data from a minimum one project. Since the data is needed from both the testers and developers, it relies on human for effort estimation. This is considered as one of the drawback of this approach. Another method has been introduced in [19] called as Accumulated Efficiency Method which also uses use case for estimation. This method is used to estimate the execution effort for testing based on the efficiency of the team. Along with the use cases, LOC is also used for software test estimation. This approach is applicable only for manually executed tests and do not apply for automated test execution. Usually small test teams prefer manual testing rather than the automated testing because of budget restrictions. The method does not require any information from historical data. Also, it does not use any natural language processing for estimation. However, estimation cannot be completed in the initial stages in development of software, because it uses LOC as one of the input. This method is also used to estimate the complete effort for testing instead of effort for single test-case execution. In order to minimize the usage of use cases for estimating the test execution effort, Nageswaran et al., proposed another method in [16] called as Function Points Estimation (FPE) method. This approach uses function points to estimate the execution effort for software testing. It requires some factors like transaction and person-hours to estimate the effort. These values are obtained from the previously executed projects of the same domain. These details are not easy to collect. Therefore, it suffers adaptive problems [17]. And also it is costly to implement. Therefore, this method is not widely used for estimation of software testing effort.

Another direction for estimating test execution is carried out in [20] called as Case-Based Reasoning (CBR). This approach used data mining techniques to classify the data. It is a reasoning approach in which previous cases are considered and reused to solve the problem of test effort estimation. This method uses data from similar cases (or situations) and hence it is difficult to identify or solve a problem if new case arrives. In such scenario, there won’t be any data to be reused to solve the problem. Even if the case is same, adopting same solution for similar problem may not yield the same result. Due to such limitations, this approach can be used only in small projects. In [21], Sharma et al., proposed the Software Requirement Specification (SRS) method. It uses SRS document to estimate the execution effort. Since this approach uses Requirement Specification document, the estimation can be done in very initial software development stage.

This method works in such a way that requirements are extracted from requirement document and complexities are mapped to them based on the level of requirements. With the assistance of the weightage alloted to the necessities dependent on multifaceted nature, test effort is evaluated. This strategy utilizes prerequisite specification rather than test specification. Therefore, it lacks clarity over test specifications. In [22], Aranha et al., overcome the above problem by using specification of test rather than specification of requirement. They created an estimation model that estimates the software test execution effort based on the test specifications. This method extracts test steps of test cases from the test specification document. It uses size and execution points for estimation the test execution effort. The points for execution are fixed based on the previous data which stores data of executed test-cases previously. For this method, test specifications should be written in a CNL. CNL is similar to that of natural language. The only difference is that the test specifications written in a CNL uses standard format (some restrictions in the usage of lexicon and grammar). In this way, there is no probability of composing an activity from numerous points of view. This methodology identifies that every single test step has one primary action word and zero or numerous contentions supporting the fundamental action word. Every single action word identifies in a test step speaks to the activity and contentions adds more data to the activity. The entire process of this approach for estimation the time execution of test is given by,

- Once the test specification file is obtained, each and every line is parsed for verb and supporting arguments
- An execution point (measure of size and execution complexity) is assigned to each and every step
- Add all the execution points obtained from the previous step
- Estimate the time used in execution of test for test-cases in man-hours

When utilizing this technique for effort estimation, experiments ought to be promptly accessible. Likewise, this strategy uses test ventures for time used in execution estimation. In this manner, the estimation expense becomes higher. Each time, the connection between test execution time and execution focuses must be demonstrated. In the proposed modal, we also utilized specifications of test for time regarding estimation, without the use of points for execution as in above method. In addition to the size of the test cases, we consider the waiting time of the test steps for test effort estimation. I also build a database that consists of previous data of test cases executed in history. Another difference with above model is that I use MT the time for execution for estimation. The MT of test cases comes from a tool called Script Editor (Section 3.1.3), where testers of LMKR fix an upper bound of time within which the test cases have to be executed. Along with the MT, I use log files from previous execution to predict AT.

2.2 Case Study Design

In addition to the literature review, a case study was also used as our research method. I communicated with LMKR testers personally and required data were collected from them. The collected data includes test execution logs. Such data are used to find the time for execution of previously executed test-cases. Based on the obtained data and the method identified from the review of literature, a modal is proposed to predict and estimate the execution time of manual test cases.

2.2.1 Research Questions

The objective of this case study is refined into a set of research questions as follows. These questions will be answered in our study analysis.

RQ1: How can the execution time of test cases can be estimated and predicted?

RQ2: How to implement and validate the execution time prediction and estimation classification?

2.2.2 Case and Subject Selection

This case study targets at finding a method for prediction and estimation of time for execution based on the test-cases that were executed before in LMKR. GeoGraphix project from LMKR is considered as a case for our study. GeoGraphix is the underground subway train used as public transportation in Stockholm, Sweden. This project is called as MOVIA C30 metro. The new fleet is running on the Red Line from northeastern Stockholm to the sub-urban part of south-west, crossing the city centre. These vehicles have driver's cab in both sides of train and also built with driver-less functionality. It satisfies environmental standards such as efficient energy usage and built using recyclable materials [23]. The test cases and test specifications obtained from this project are used as the input for our case study.

2.2.3 Data Collection Procedures

The data collection involves the use of direct method [24] and collecting di.e., personal communication (informal interviews) with LMKR Domain Testing team. In other words, the subjects are contacted directly and the real test cases are collected. This collected data acts as the input/starting point of the research.

2.2.4 Analysis Procedures

The time mentioned in the Test Manager tool (Section 3.1.3) and log files are the main source for our analysis procedure. The test case also contains time such as WT. These times are utilized for our algorithm. MT, WT plays a major role in the execution time estimation. There are the cases where the test case does not contain specific times. Such cases will use our database. The database is searched across the 'verb' and 'arguments' for the execution time (Section 3.1.3).

2.2.5 Validity Procedures

The validity of collected data is verified throughout the phases of case study. The collected data is analyzed to define a prototype/algorithm to predict and estimate the execution time. Once it has been done, the data is validated in the implemented prototype. The execution time from the proposed solution is validated with the actual execution time of the selected test cases.

CHAPTER 3: DATASET COLLECTION

3.1 Introduction

This chapters provides the brief overview of the term Data Set or Dataset. More on detailed information on the Test Cases and their complexities. The procedures to get the data or create the datasets for personal use or commercial use. The TFS repository to gather the datasets for the analysis of the proposed models and research methods.

3.1.1 Dataset

A collection of the Data and information is called the Dataset. Most commonly the dataset is representing a single relation of the database. The data set contains the properties for every object, like height and weight. Each value is known as a datum. The dataset may consist of data for one or more than one members that corresponds to the number of rows [25].

3.1.2 Manual Testing Dataset

The datasets of test cases consists of complex and normal test cases.

Table 3-1: Attributes for Test-Case DataSet

Name	Description	Type
TestCaseID	An identifier for a test-case	Number
TestCaseClassCoverage	The class number covered by the test case. We choose 2 classes, that range from 1-2	Number
TestCasePathCoverage	The paths covered in the test-case	Number
TestCaseMethodCoverage	The methods covered in the test-case	Number
TestCaseBrancheCoverage	The branches covered in the test-cases.	Number
TestCaseRunTimeDurartion	The time required for execution of test-case	Number
TestCaseInput	The test-case inputs	varchar
TestCaseOutput	The test-case expected outputs	varchar
TestCaseStatus	An Indicator to represent the test-case faults or not. If the status shows fail, it mean that fault is detected	T/F [0, Fail] [1, Pass]

TestCaseCycomaticComplexity	The cyclomatic complexity for the test-case	Number
TestCaseSteps	The steps for execution of test-case	Number
TestCaseClassLabel	An identifier of Classlabel	Normal=0 Complex=1

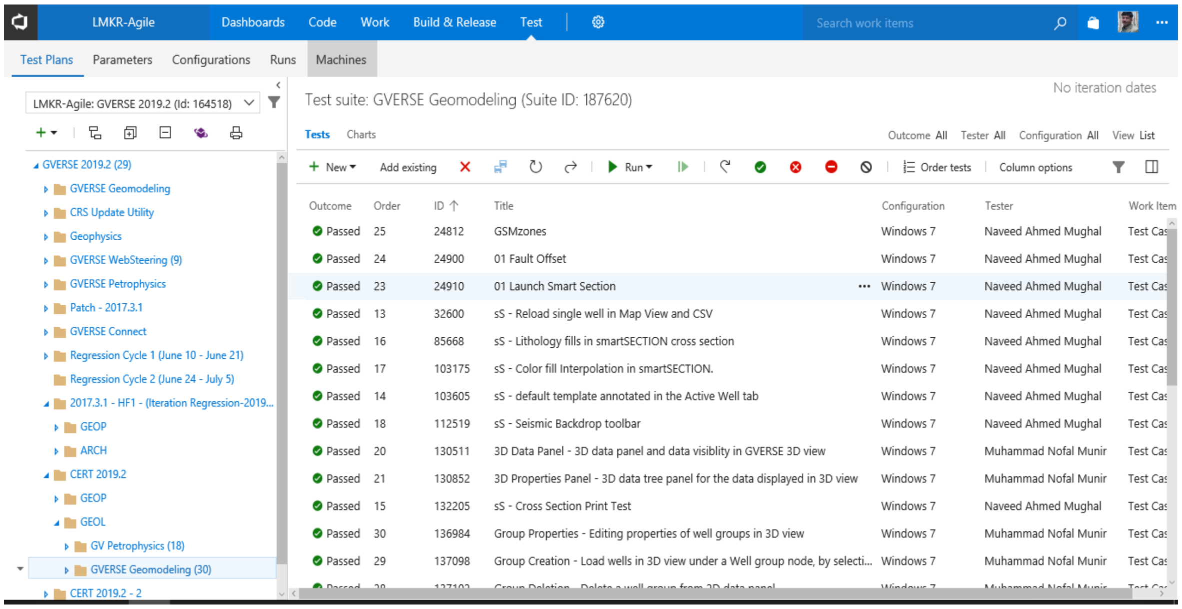
3.2 Dataset Collection Methods

3.2.1 Interviews and Questioners

The Dataset collection and data mining techniques are directly connected to each other. Most simple and traditional way to create a dataset is to create the questionnaires and ask questions manually. Manual collection methods for example paper based and electronic survey are used to creation of the test-cases, but the dataset collected in such a manner can contain biasness and uninterested behavior of the persons.

3.2.2 Test Manager

The data is extracted from Test Manager Database by exporting the required attributes of the Test Case work item. Through forums the data that can be collected is limited in the size and quantity.



Test suite: GVERSE Geomodeling (Suite ID: 187620)

Outcome	Order	ID	Title	Configuration	Tester	Work Item
Passed	25	24812	GSMzones	Windows 7	Naveed Ahmed Mughal	Test Cas
Passed	24	24900	01 Fault Offset	Windows 7	Naveed Ahmed Mughal	Test Cas
Passed	23	24910	01 Launch Smart Section	Windows 7	Naveed Ahmed Mughal	Test Cas
Passed	13	32600	sS - Reload single well in Map View and CSV	Windows 7	Naveed Ahmed Mughal	Test Cas
Passed	16	85668	sS - Lithology fills in smartSECTION cross section	Windows 7	Naveed Ahmed Mughal	Test Cas
Passed	17	103175	sS - Color fill Interpolation in smartSECTION.	Windows 7	Naveed Ahmed Mughal	Test Cas
Passed	14	103605	sS - default template annotated in the Active Well tab	Windows 7	Naveed Ahmed Mughal	Test Cas
Passed	18	112519	sS - Seismic Backdrop toolbar	Windows 7	Naveed Ahmed Mughal	Test Cas
Passed	20	130511	3D Data Panel - 3D data panel and data visiblity in GVERSE 3D view	Windows 7	Muhammad Nofal Munir	Test Cas
Passed	21	130852	3D Properties Panel - 3D data tree panel for the data displayed in 3D view	Windows 7	Muhammad Nofal Munir	Test Cas
Passed	15	132205	sS - Cross Section Print Test	Windows 7	Naveed Ahmed Mughal	Test Cas
Passed	30	136984	Group Properties - Editing properties of well groups in 3D view	Windows 7	Muhammad Nofal Munir	Test Cas
Passed	29	137098	Group Creation - Load wells in 3D view under a Well group node, by selecti...	Windows 7	Muhammad Nofal Munir	Test Cas

Figure 3.1 Test Cases in Test Manager

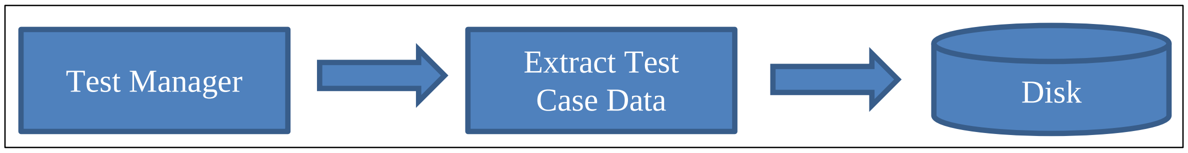


Figure 3.2 Data Process Diagram

CHAPTER 4: PROPOSED MODEL FOR ESTIMATION USING CLASSIFIER

Our strategy to estimating the test-case effort comprises of three steps. We begin by getting the test-cases for the system being tested and producing them. Then we construct our test-case dataset based on the test-case's most significant attribute. Finally, we recognize the complex test-case, pick the classifier and rules to be applied and extract the outcomes for this. Below we describe each of these steps in detail.

4.1 Proposed Model

Our Proposed framework is to detect the complex Test cases then apply the classification methods for the evaluation.

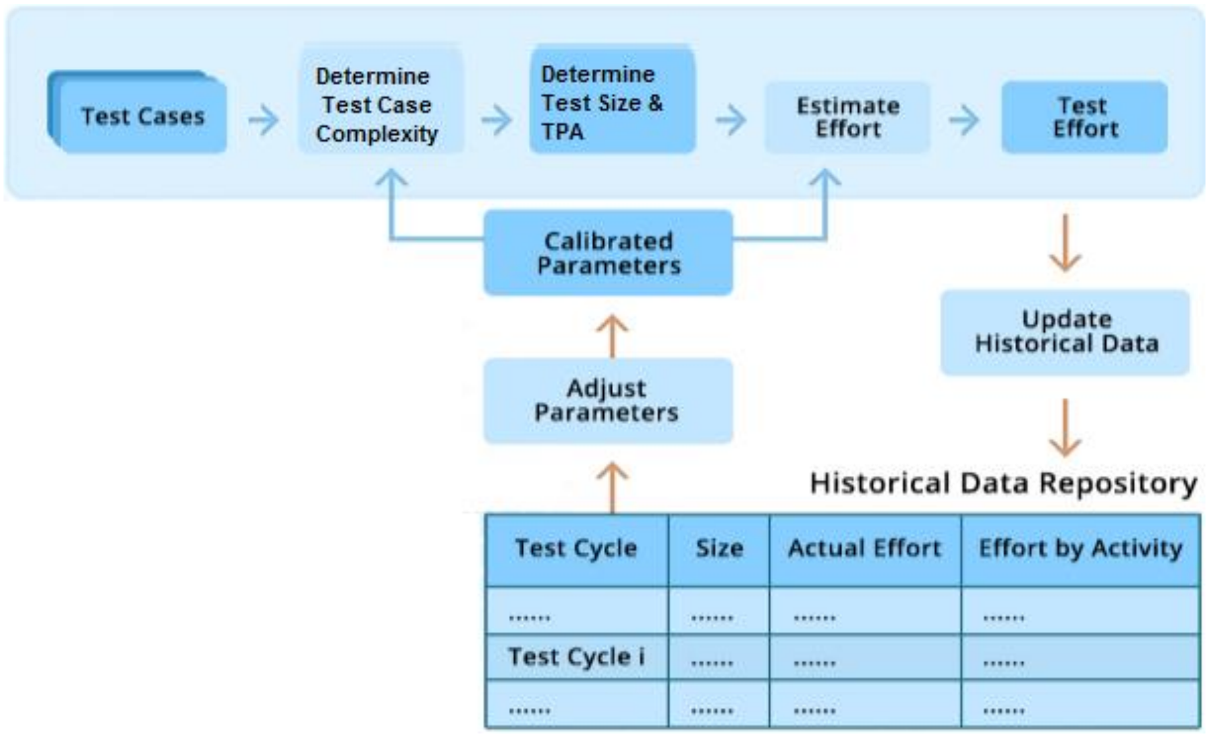


Figure 4.1 Proposed Model

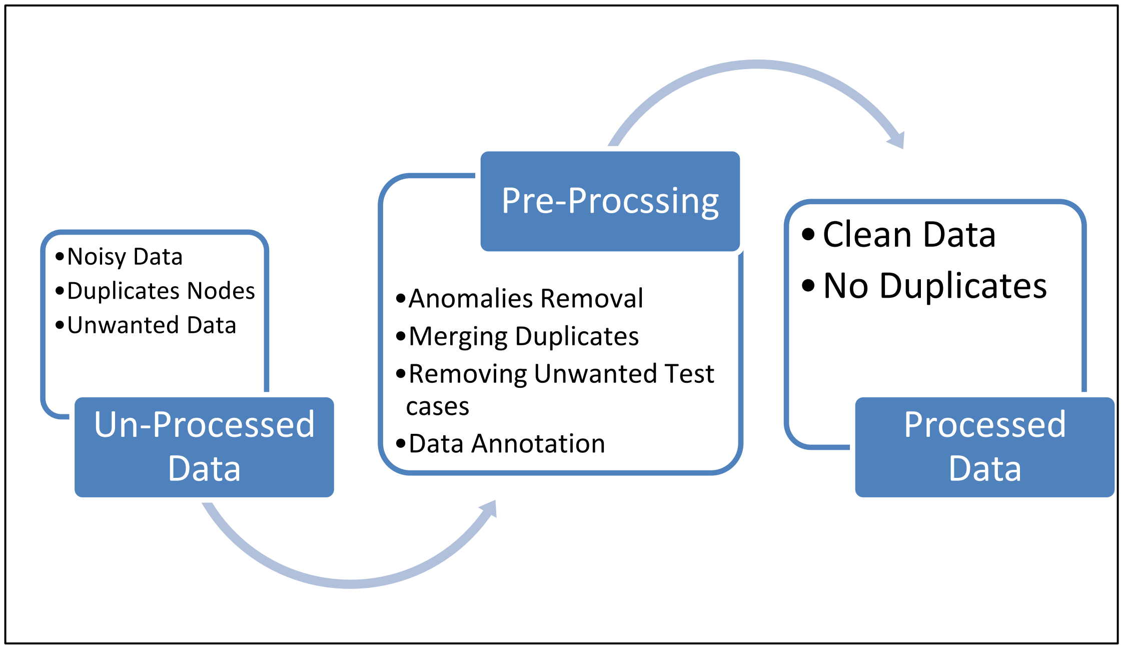
4.2 Test-Cases

A TEST-CASE is a set of conditions or variables under which a tester will determine whether a test scheme meets or operates properly. The method of creating test-cases can also assist to identify issues in an application at requirement or design phase.

4.3 Pre-Processing of Data

Data pre-processing is essential part of the Time estimation Analysis. Data sets are often noisy. This need to handle data issue before classification as data set is required to test and train our model and unclean data can affect classifier results. Pre-processing of data have several steps some of them are removal of ambiguities and anomalies, either merge or remove the same nodes to remove data repetition and establish pillars in multi-layer structure. Pre-processing also includes the transformation of the data into desired format and saving the required file at desired location.[27].

Figure 4.2 Pre-processing flow



4.3.1 Anomalies Removal

4.3.1.1 Missing Values

- Remove the values with the missing terms
  - Removal should not delete more than the 6% of the Data
- Fill the missing values
  - Either replace with frequent or average value
- Use the predicting algorithm to predict the missing values
  - Decision Tree
  - Classification Model
  - Regression

4.3.1.2 Aggregation

Purpose:

The Purpose of the aggregation is to remove the irrelevant features and merge the same attributes. For example date and time can be merged, similarly Month, day and year can be combined to create the single attribute. Aggregation provides the following benefits:

- Reduction of the Data
- Abstract view
- Stable data

4.3.1.3 Irrelevant features

Irrelevant features, we mean sometime the data contain such information that is of no use in the analysis and visualization. Several attributes have information that is not useful in the analysis and prediction e.g. Students' ID is often irrelevant to the task of predicting student's grades

4.3.2 Merging Duplicates

In Test suite there is one Test case can have more than one similar Test case steps.

4.3.3 Removing Unwanted

The Dataset collected by anyways can have various anomalies. The Dataset can have the attributes that are not relevant to our research. The Dataset can have various steps that are not reliable or can be fake.

4.4 Classification and Prediction

Classification of data is very popular in machine-learning technique. Classification uses the given input to predict the outcome. Classification is used to solve out the wide range of the problems i.e. either simple or complex problems [28] [29].

4.4.1 Classifiers

Classifiers use the phenomenon of training and testing. They divide the provided data into two disjoint sets i.e. training and testing subsets. f(x) =fxtr + fxte Training fxtr and testing fxte subsets contains the objects of the both classes i.e. (Complex and Normal). The objects in these subsets are added randomly, so that the subsets are biasness free. The selected classifier is trained using the subset fxtr for training and performance of the model is evaluated by testing the subset fxte.

4.4.1.1 Naïve Bayes

Naive Bayes is a very high biased, very low in variance classifier, and it can generate a perfect model even with a very-small dataset. It is very simple in use and very inexpensive for computation. Normally used for text categorization, spam-detection, sentiment-analysis, and recommender systems.

4.4.1.2 Decision Tree

Decision-tree is a tool for making decision that apply graph like a tree for prediction of possible results. In Decision Tree we divide the complete dataset into smaller subsets sets and generates decision tree. The main algorithm used for decision trees is known as ID3 created by J. R. Quinlan.

4.4.2 Cross Validation

Cross validation is process which is used to estimate and evaluate the performance of a created and designed model. The operator that is mostly for Cross-validation is mostly used for testing the performance of the operator that was trained used in the practice.

Cross Validation is process that is dependent on two sub processes. These two processes are called the testing sub phase and the training sub phase. During the training phase on the labeled dataset the model is trained. Then this trained model is applied in the testing phase. The results of the testing phase determines the performance of the trained model.

Example Set is divided into K subsets of equal sizes. The K-1 subsets are than used for the training and the remaining one is used for the testing of the trained model. The process is repeated K many times such that the data once used for the testing is not tested again. The results from all the iterations are combined or either averaged to produce the single output estimation.

The trained model may perform well on the unlabeled data of same training and testing dataset. This means that the model may perform well on the testing data, but for the unseen and generalize data the model may be worse than the testing outcomes.

4.4.3 Voting

Operator of Rapid Minor used for voting to combine the prediction power used for more than one classifiers to attain better results than the results of the only one classifier was used.

Rather than training one classifier two classifiers were trained to predict the possible outcome, these classifier are Cdt, CNaive.

For each test example v_i, a_i is computed for n classifiers trained using feature vector f_{vi}.Voting methods depending on majority prediction finally predict 1 (Complex) or 0 (normal testcase).

$$\boldsymbol{a = vote(a_{i(c)}, a_{i(c+1)}, .., a_{i(c+n)})}$$

Where

$$a_{i(c)=}C_{knn}(f_{vi}),$$
$$a_{i(c+1)=}C_{dt}(f_{vi}),$$
$$a_{i(c+n)=}C_{svm}(f_{vi}).$$

It is possible to classify testing operations into four major categories, test planning, test design, test execution, and defects reporting. During the course of the project, test execution and reporting of defects may be carried out multiple times for a single test case. However, all of these operations are taken into consideration by the size measured in Test Case Point, assuming that each activity is carried out once. It is possible to generate the allocation of testing effort using historical information. The distribution of test phase effort shown in Table 6 was achieved from the same study conducted to collect the above-mentioned constants. Again, these values represent primarily the experience in LMKR and that was encouraged to modify these figures using their own or previous data.

4.5 Estimation

Activities for testing can be categorized into four areas, test organization, test arrangement, test execution, and flaw declaring. Of these activities, test execution and disfigurement enumerating may be played out various events for a lone investigation during endeavor. In any case, the size evaluated in Test Case Point thinks about these activities, tolerating that each activity is performed once. The assignment of effort used in these activities licenses assessing the effort used in case the test-execution & flaw uncovering activities are performed more than once. The scattering of testing effort can be made using chronicled data. The scattering of effort of testing stages showed up in Table 6 was gained from a comparative diagram performed to assemble the constants depicted beforehand. Afresh, these characteristics generally reflect the contribution in target affiliation, and consequently, the evaluators are encouraged to change the numbers based on their own understanding or recorded data previously.

CHAPTER 5: RESULTS AND DISCUSSION

This Chapter of the Research includes results and analysis of the proposition which is identification of the important test cases in the testing.

5.1 Building Dataset

The data sets of our research have several features. Table 5 has all of these attributes listed. Several plugins and tools have been used to get the values of the mentioned attributes. For instance to measure the values of coverage for class, path, method; Eclemma [20] a plugin in Eclipse has been used. We have used Team Explorer[21] which is also a plugin in Visual Studio to calculate runtime duration required for each test case. Lastly we use Test Manager [20] to calculate cyclomatic complexity. There are total 493 test cases in the data set with a total of 10 attributes and Class labels. The normal test cases are 268 whereas 225 are complex test cases indicating a balanced dataset. Lastly we have provided the inputs outputs and status of each test case. We have used Replace Missing Value Filter to substitute the missing value by mean value of other instances in the same attribute. There were many proposed tools including RAPID MINER, WEKA, & ORANGE however we chose RAPID MINER for this purpose because of its higher accuracy and as well as the fact that it has different types of classifiers. When applied to different data sets Naïve Bayes classifier is the best classifier because of its accuracy [22]. In order to generate different patterns and set of rules we have applied machine learning techniques on Rapid Miner which include Naïve Bayes and J48 Decision Tree for deciding whether a test case is normal or complex. As described in Section 2 the following are the rules used to predict whether or not a test case is complex:

5.1.1 Pre-processing Data

The data collected by the crawler is preprocessed before apply any of the graph theory. The Missing values were handled, various nodes have been merged as they were pointing towards the same person (Node). The Data is then visualized and analyzed through Gephi Visualization software.

5.2 Classification

5.2.1 Rapid Miner

Rapid miner is an open source software tool that provides the users the platform to train and test the classification models for various applications [33]. We have used the Rapid miner to test and train the proposed hybrid classifier to predict the complex test cases.

5.2.2 Rapid Minor Process

The Figures below depicts the model process of the classification. The Figure 1 (Appendix) shows the main and level 0 of the Process having Dataset Retrieval and Cross Validation connected to Output. While Figure 2 (Appendix) show the level 1 of the process that is inside of the cross validation operator. It contains the operators to measure the performance and prediction classifiers. And the 2nd level of the process is the Voting operator which combines the generated result of all three classifiers is shown in Figure 3 (Appendix).

5.2.3 Classification

The detailed results of the proposed schema is depicted in the following section. The performance rating and throughput of proposed schema is calculated using various measures, these measures include:

- Sensitivity
- Specificity
- Accuracy
- (ROC) curves (AUC)

These measures are calculated using Eq. (6.1), Eq. (6.2) and Eq. (6.3) respectively.

$$\text{Sensitivity} = \frac{\text{TP}}{(\text{TP} + \text{FN})} \tag{6.1}$$

$$\text{Specificity} = \frac{\text{TN}}{(\text{TN} + \text{FP})} \tag{6.2}$$

$$\text{Accuracy} = \frac{(\text{TP} + \text{TN})}{(\text{TP} + \text{FP} + \text{FN} + \text{TN})} \tag{6.3}$$

- TP are the number of the complex test cases that are identified correctly by the model known as true positive
- TN is called the true negatives. It is the number of the normal nodes that are correctly identified.
- FP means false positives, it defines the number of the normal nodes that are identified as the complex test cases.
- FN means False Negatives, it illustrates the number of the complex test cases that are identified as the normal during the classification phase.

The trained models are tested with the help of the cross validation. The Value of K is selected in such a way that for training about 70% of data was used as and remaining data used for testing which is 30%. The experimental procedures are repeated K times and their average or combined results are given. Table below illustrates the results of proposed paradigm for complex test cases detection on all projects given in case studies.

Table 5-1: Performance Review of proposed modal for Test-Case complexity detection

Case study	Sensitivity	Specificity	Accuracy
I	90.91%	99.76%	99.52%
II	100.00%	99.39%	99.40%

The classifier which is hybrid is compared with individual, Naïve Bayes and Decision Tree classifiers. Below Table 5-6 compares all these in terms of accuracy for complex test case detection.

Table 5-2: Classifier Comparison

Method	**Case Study - 1**	**Case Study – 2**

Naive Bayes	94.15%	96.97%
Decision Tree	65.79%	78.34%

The proposed modal has been tried utilizing two contextual analyses and number of factual measures. The outcomes taken unmistakably demonstrate the legitimacy and accuracy of proposed modal. Another examination from real nearby occasion is taken and proposed framework is tried on that also.

This thought recognized experiments with accuracies of 79%, 83.33% for both cases individually. The proposed modal accomplished accuracies of 96.73%, 99.59% for same contextual investigations separately. The outcomes demonstrated the legitimacy of our structure and it very well may be utilized for location complex experiments for any area.

5.3 Training and Testing

In This section we have applied the several classifiers i.e. Naïve Bayes and Decision tree further more we also applied the hybrid classifier to predict the roles of the nodes in an unlabeled and unseen dataset for the hybrid classification. The logical diagram below depicts the process followed to achieve the results. The reference screenshots of the whole process is also available in the Appendix section.

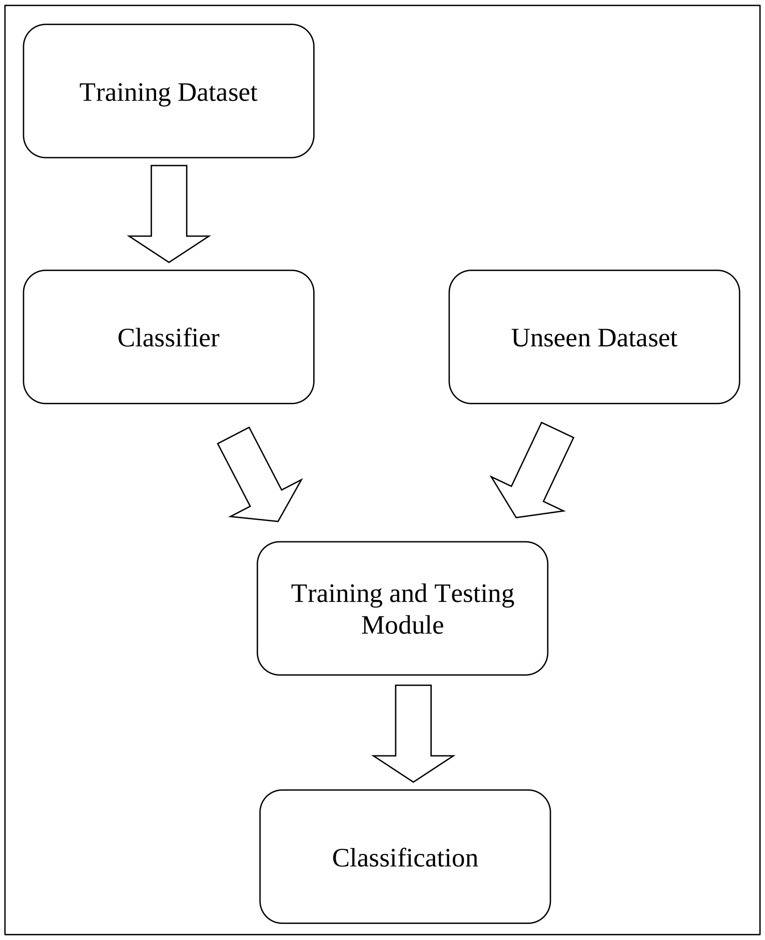


Figure 5.10 Classification Process on Unseen Data

5.3.1 Naïve Bayes

Naïve Bayes Classifier is used in the above figure for the testing and training. The Table Below depicts the results obtained by the classification. We have used Case-I for the training and Case-II for the testing.

Decision Tree Classifier is used in the above figure for the testing and training. The Table Below depicts the results obtained by the classification. We have used Case-I for the training and Case-II for the testing.

CHAPTER 6: CONCLUSION AND FUTURE WORK

In this chapter we have concluded all the results and evaluated the contributions that are made in the field of the business. The latest information and technologies are used to improve the business. We have also provided the direction to carry on the research in this domain

6.1 Conclusion

In this thesis report, we have presented system for time of execution estimation for experiments. Programming testing assumes a significant job in the achievement of programming advancement and upkeep ventures. Assessing testing exertion precisely is a key advance towards to that objective. While trying to plug the gap in assessing programming testing, this paper has suggested a strategy known as Test Case Analysis to gauge and figuring the size and exertion of programming testing exercises. The contribution of this investigation is experiments, and the yield is the quantity of Test-Case Points for the experiments being tallied.

An invaluable component of this methodology is that it quantifies the unpredictability of experiment, the fundamental work item that the analyzer creates and utilizes for test execution. Therefore, it better copies the exertion that the analyzer spends on their exercises. Another preferred perspective is that the investigation could be accomplished effectively through tallying the quantity of checkpoints, estimating the multifaceted nature of precondition and test information, and deciding the sort of each test-case.

One is the estimation algorithm, which is used to time for execution estimation for maximum time to execute the test-cases a system takes. The other one is the prediction algorithm which is used for prediction of actual time required to execute a test case with the help of its estimated maximum time. Using our proposed algorithms, we are able to predict and estimate the maximum and actual time required to execute the manual test case from integration testing. We used Test case work Items as an input for the algorithm. Initially we have created a database that has been captured from different sources namely Test case work Items, log files and Test Manager tool. The database is build for the previously executed test-cases. By utilizing the database, our algorithm estimates the MT and predicts the AT. A case study has been conducted in LMKR where both the algorithms are implemented and verified. The results are validated with the logs from the system in which the test cases are executed.

6.2 Future Works

As a future work, we propose that this approach has to be fine-tuned in such a way the predicted time should be close to the actual time. In reality, execution time for manually executing the test cases are dependent on some other factors like characteristics of system, in which it is executed and also the skills of testers. In our results section, it has been already seen that the execution time varies within a system.

In any case, there are a few constraints of this methodology, consequently recommending headings for future enhancements of the methodology. One impediment is that the Test-Case Point measure has not been exactly approved. Information of the past testing undertakings of different areas should be utilized to approve the impact and convenience of the size measure in assessing the exertion of the product test-case. Another constraint is the worry of whether the multifaceted nature scopes of the experiment's precondition and test information can appropriately mirror the real intricacy of these traits. Future enhancements for the strategy need to address these confinements.

With respect to the testers' skills, an experienced tester will take less time to execute a test-case when compared with an inexperienced tester. These factors should also be considered during prediction and estimation work in future.

REFERENCES

[1] V. Casey, Software Testing and Global Industry: Future Paradigms, Cambridge Scholars Publisher, 2008.

[2] W. Afzal, S. Alone, K. Glocksien, R. Torkar, Software test process improvement approaches, Journal of Systems and Software 111 (C) (2016) 1–33.

[3] J. Itkonen, M. V. Mantyla, C. Lassenius, Defect detection efficiency: Test case based vs. exploratory testing, in: First International Symposium on Empirical Software Engineering and Measurement, 2007, pp. 61–70.

[4] E. Dustin, T. Garrett, B. Gauf, Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality, Pearson Education, 2009.

[5] P. E. Strandberg, D. Sundmark, W. Afzal, T. J. Ostrand, E. J. Weyuker, Experience report: Automated system level regression test prioritization using multiple factors, in: 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), 2016.

[6] D. Flemstro¨m, P. Potena, D. Sundmark, W. Afzal, M. Bohlin, Similarity-based prioritization of test case automation, Software Quality Journal.

[7] V. Garousi, M. Felderer, M. Kuhrmann, K. Herkilog˘lu, What industry wants from academia in software testing? hearing practitioners' opinions, in: Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, ACM, 2017, pp. 65–69.

[8] W. Afzal, A. N. Ghazi, J. Itkonen, R. Torkar, A. Andrews, K. Bhatti, An experiment on the effectiveness and efficiency of exploratory testing, Empirical Software Engineering 20 (3) (2015) 844–878.

[9] J. Kasurinen, O. Taipale, K. Smolander, Test case selection and prioritization: Risk-based or design-based?, in: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, 2010, pp. 10:1–10:10.

[10] S. Tahvili, W. Afzal, M. Saadatmand, M. Bohlin, D. Sundmark, S. Larsson, Towards earlier fault detection by value-driven prioritization of test cases using fuzzy topsis, in: 13th International Conference on Information Technology : New Generations, 2016.

[11] E. Engstr¨om, P. Runeson, Decision support for test management and scope selection in a software product line context, in: 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, 2011, pp. 262–265.

[12] M. Felderer, R. Ramler, Integrating Risk-based Testing in Industrial Test Processes, Vol. 22, Kluwer Academic Publishers, 2014, pp. 543– 575.

[13] P. E. Strandberg, W. Afzal, D. Sundmark, Decision making and visualizations based on test results, in: 12th International Symposium on Empirical Software Engineering (ESEM), 2018.

[14] S. Singh, F. Sahib, Optimized test case prioritization with multi criteria for regression testing, in: International Journal of Advanced Research in Computer Engineering & Technology, 2014.

[15] Z. Li, M. Harman, R. M. Hierons, Search algorithms for regression test case prioritization, IEEE Transactions on Software Engineering 33 (4) (2007) 225–237.

[16] D. Hao, L. Zhang, H. Mei, Test-case prioritization: achievements and challenges, Frontiers of Computer Science 10 (5) (2016) 769–777.

[17] S. Tahvili, M. Saadatmand, S. Larsson, W. Afzal, M. Bohlin, D. Sundmark, Dynamic integration test selection based on test case dependencies, in: The 11th Workshop on Testing: Academia-Industry Collaboration, Practice and Research Techniques, 2016.

[18] S. Tahvili, A Decision Support System for Integration Test Selection, 2016.

[19] S. Tahvili, M. Saadatmand, M. Bohlin, W. Afzal, S. H. Ameerjan, Towards execution time prediction for test cases from test specification, in: 43rd Euromicro Conference on Software Engineering and Advanced Applications, 2017.

[20] Garousi, M. Felderer, J. a. M. Fernandes, D. Pfahl, M. V. Ma¨ntyl¨a, Industry-academia collaborations in software engineering: An empirical analysis of challenges, patterns and anti-patterns in research projects, in: Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, 2017, pp. 224–229.

[21] W. Afzal, R. Torkar, Incorporating metrics in an organizational test strategy, in: 2008 IEEE International Conference on Software Testing Verification and Validation Workshop, 2008.

[22] L. Angelis, I. Stamelos, M. Morisio, Building a software cost estimation model based on categorical data, in: Proceedings Seventh International Software Metrics Symposium, 2001, pp. 4–15.

[23] S. Nageswaran, Test effort estimation using use case points, Quality Week (2001) 1–6.

[24] X. Zhu, B. Zhou, F. Wang, Y. Qu, L. Chen, Estimate test execution effort at an early stage: An empirical study, in: International Conference on Cyberworlds, 2008.

[25] P. R. Srivastava, A. Varshney, P. Nama, X.-S. Yang, Software test effort estimation: A model based on cuckoo search, Int. J. Bio-Inspired Comput. 4 (5) (2012) 278–285.

[26] D. G. e. Silva, B. T. de Abreu, M. Jino, A simple approach for estimation of execution effort of functional test cases, in: Second International Conference on Software Testing Verification and Validation, 2009.

[27] E. R. C. de Almeida, B. T. de Abreu, R. Moraes, An alternative approach to test effort estimation based on use cases, in: 2009 International Conference on Software Testing Verification and Validation, 2009, pp. 279–288.

[28] A. Sharma, D. S. Kushwaha, An empirical approach for early estimation of software testing effort using srs document, CSI Transactions on ICT 1 (1) (2013) 51–66.

[29] A. Sharma, D. S. Kushwaha, Complexity measure based on requirement engineering document and its validation, in: 2010 International Conference on Computer and Communication Technology (ICCCT), 2010, pp. 608–615.

[30] E. Aranha, P. Borba, An estimation model for test execution effort, in: First International Symposium on Empirical Software Engineering and Measurement, 2007.

[31] R. Torkar, N. Awan, A. Alvi, W. Afzal, Predicting software test effort in iterative development using a dynamic bayesian network, in: Proceedings of the 21st International Symposium on Software Reliability Engineering - Industry Track, 2010.

[32] V. Nguyen, V. Pham, V. Lam, qestimation: A process for estimating size and effort of software testing, in: Proceedings of the 2013 International Conference on Software and System Process (ICSSP'13), 2013.

[33] ISO/IEC/ STANDARD IEEE 29119-1, Software and systems engineering — Software testing — Part 1: Concepts and definitions, Standard, ISO/IEC/IEEE (2013).

[34] A. Bush, G. Baladi, A. C. D.-. on Road, P. Materials, A. C. D.-. on Soil, Rock, Nondestructive Testing of Pavements and Backcalculation of Moduli, no. no. 1026 in ASTM STP 1026, ASTM, 1989.

[35] R. Revlin, Cognition: Theory and Practice, Worth Publishers, 2012.

[36] A. Knott, Sensorimotor Cognition and Natural Language Syntax, MIT Press, 2012.

[37] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, P. Stenstro¨m, The worst-case execution time problem - overview of methods and survey of tools, ACM Trans. Embed. Comput. Syst. 7 (3) (2008) 36:1–36:53.

[38] P. Refaeilzadeh, L. Tang, H. Liu, Cross-Validation, Springer US, Boston, MA, 2009, pp. 532–538.

[39] F. Harrell, Regression Modeling Strategies: With Applications to Linear Models, Logistic Regression, and Survival Analysis, Graduate Texts in Mathematics, Springer, 2001.

[40] J. H. Friedman, C. B. Roosen, An introduction to multivariate adaptive regression splines, Statistical Methods in Medical Research 4 (3) (1995) 197–217.

[41] Y. Chang, C. Hsieh, K. Chang, M. Ringgaard, C. Lin, Training and testing low-degree polynomial data mappings via linear svm, Journal of Machine Learning Research 11 (2010) 1471–1490.

[42] D. Dipayan, Deep learning with Hadoop: build, implement and scale distributed deep learning models for large-scale datasets, Birmingham, UK, 2017.

[43] S. Bird, E. Klein, E. Loper, Natural Language Processing with Python, O'Reilly, 2009.

[44] P. Runeson, M. Ho¨st, Guidelines for conducting and reporting case study research in software engineering, Empirical Software Engineering 14 (2) (2008) 131.

[45] E. Engstro¨m, P. Runeson, A. Ljung, Improving regression testing transparency and efficiency with history-based prioritization – an industrial case study, in: 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation, 2011, pp. 367–376.

[46] BOMBARDIER, Bombardier wins order to supply new generation movia metro fleet for stockholm, Tech. rep. (2017).

[47] S. Tahvili, Polynomial and spline regression analysis, https://github. com/sahar82/JSS (2018).

[48] J. Devore, Probability and Statistics for Engineering and the Sciences, Cengage Learning, 2011.

[49] K. R. Muller, S. Mika, G. Ratsch, K. Tsuda, B. Scholkopf, An introduction to kernel-based learning algorithms, IEEE Transactions on Neural Networks 12 (2) (2001) 181–201.

[50] A. K. Jain, R. P. W. Duin, J. Mao, Statistical pattern recognition: A review, IEEE Trans. Pattern Anal. Mach. Intell. 22 (1) (2000) 4–37.

[51] C. Chen, Y. Wang, Y. Chang, K. Ricanek, Sensitivity Analysis with Cross-Validation for Feature Selection and Manifold Learning, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 458–467.

[52] W. Afzal, R. Torkar, R. Feldt, Resampling methods in software quality classification, International Journal of Software Engineering and Knowledge Engineering 22 (02) (2012) 203–223.

[53] H. Martens, M. Martens, Multivariate analysis of quality. an introduction, Measurement Science and Technology 12 (10) (2001) 1746.

[54] E. L. G. Alves, P. D. L. Machado, T. Massoni, S. T. C. Santos, A refactoring-based approach for test case selection and prioritization, in: 8th International Workshop on Automation of Software Test (AST), 2013, pp. 93–99.

[55] W. Guang, M. Baraldo, M. Furlanut, Calculating percentage prediction error: A user's note, Pharmacological Research 32 (4) (1995) 241 – 248.

[56] C. Robson, Real world research : a resource for users of social research methods in applied settings, third edition Edition, Chichester, West Sussex John Wiley & Sons, 2011.

[57] C. Wohlin, P. Runeson, M. Ho¨st, M. C. Ohlsson, B. Regnell, A. Wessl´en, Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers, 2000.

[58] P. Cozby, C. Rawn, Methods in Behavioural Research, McGraw-Hill Ryerson, 2012.

[59] E. A. Drost, Validity and reliability in social science research, Education, research and perspectives. 38 (1).

[60] S. Tahvili, M. Saadatmand, M. Bohlin, Multi-criteria test case prioritization using fuzzy analytic hierarchy process, in: The Tenth International Conference on Software Engineering Advances, 2015.

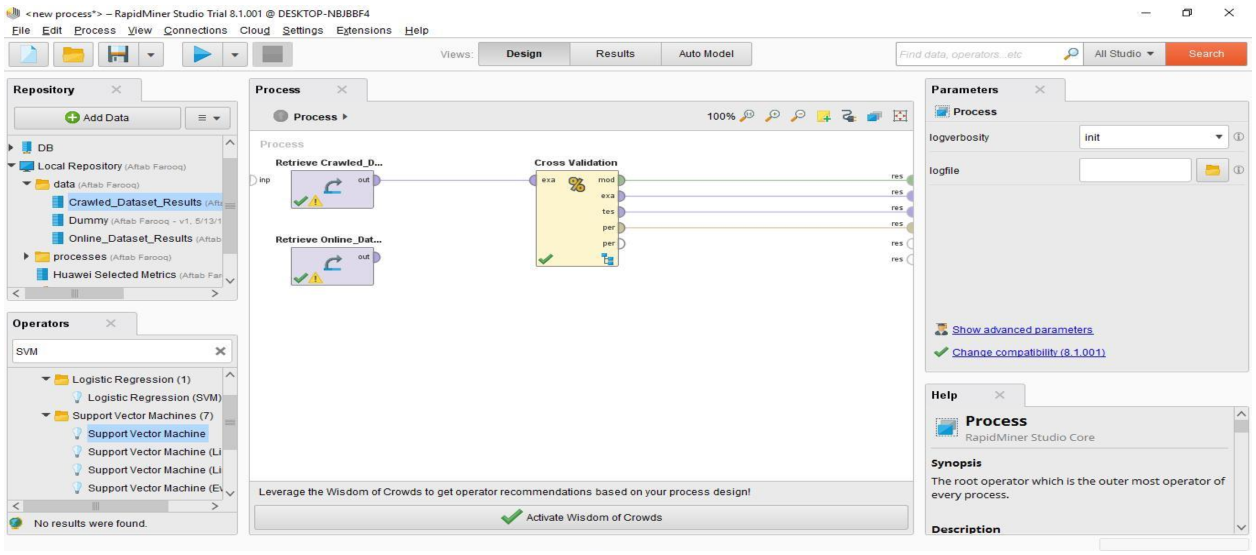
[61] J. Hipp, U. Gu¨ntzer, G. Nakhaeizadeh, Algorithms for association rule mining - a general survey and comparison, SIGKDD Explor. Newsl. 2 (1) (2000) 58–64.

[62] H. Yamamoto, Network system, server, client terminal, timeout information providing method, timeout information display method, and programs, sA Patent 8495222 (Jul. 23 2013).

[63] T. G. Dietterich, Approximate statistical tests for comparing supervised classification learning algorithms, Neural Comput. 10 (7) (1998) 1895– 1923.

APPENDIX

Below are the image representation of the created classification model in the Rapid Minor. These images are very helpful to set up the workflow.



<new process*> – RapidMiner Studio Trial 8.1.001 @ DESKTOP-NBJBBF4
File Edit Process View Connections Cloud Settings Extensions Help
Views: Design Results Auto Model
All Studio Search
Repository
Add Data
DB
Local Repository (Aftab Farooq)
data (Aftab Farooq)
Crawled_Dataset_Results
Dummy (Aftab Farooq - v1, 5/13/1
Online_Dataset_Results
processes (Aftab Farooq)
Huawei Selected Metrics
Operators
SVM
Logistic Regression (1)
Logistic Regression (SVM)
Support Vector Machines (7)
Support Vector Machine
Support Vector Machine (Li
Support Vector Machine (Li
Support Vector Machine (Ev
Process
Process
100%
Retrieve Crawled_D...
Retrieve Online_Dat...
Cross Validation
Parameters
Process
Show advanced parameters
Change compatibility (8.1.001)
Help
Process
RapidMiner Studio Core
Activate Wisdom of Crowds

No results were found.

logverbosity: init

logfile:

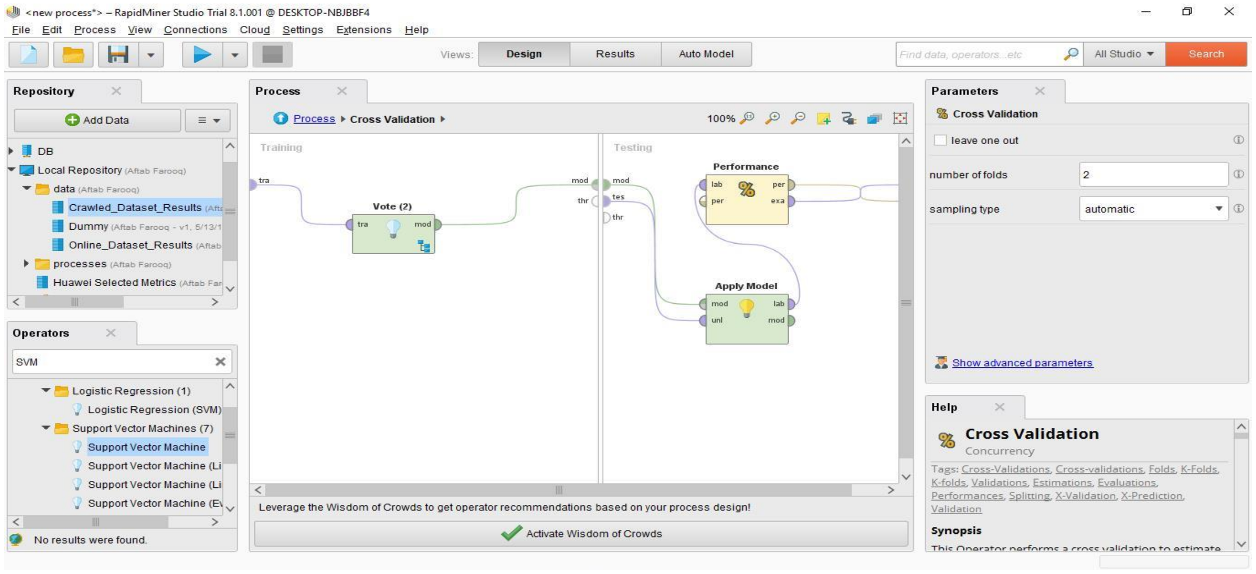
Synopsis

The root operator which is the outer most operator of every process.

Description

Leverage the Wisdom of Crowds to get operator recommendations based on your process design!

Figure 1. Level 0 Classification Process Diagram



<new process*> – RapidMiner Studio Trial 8.1.001 @ DESKTOP-NBJBBF4
File Edit Process View Connections Cloud Settings Extensions Help
Views: Design Results Auto Model
All Studio Search
Repository
Operators
SVM
Process
Process ▸ Cross Validation ▸
Training
Testing
Vote (2)
Performance
Apply Model
Parameters
Cross Validation
Show advanced parameters
Help
Cross Validation
Concurrency
Activate Wisdom of Crowds

No results were found.

leave one out: ☐

number of folds: 2

sampling type: automatic

Tags: Cross-Validations, Cross-validations, Folds, K-Folds, K-folds, Validations, Estimations, Evaluations, Performances, Splitting, X-Validation, X-Prediction, Validation

Synopsis

This Operator performs a cross validation to estimate

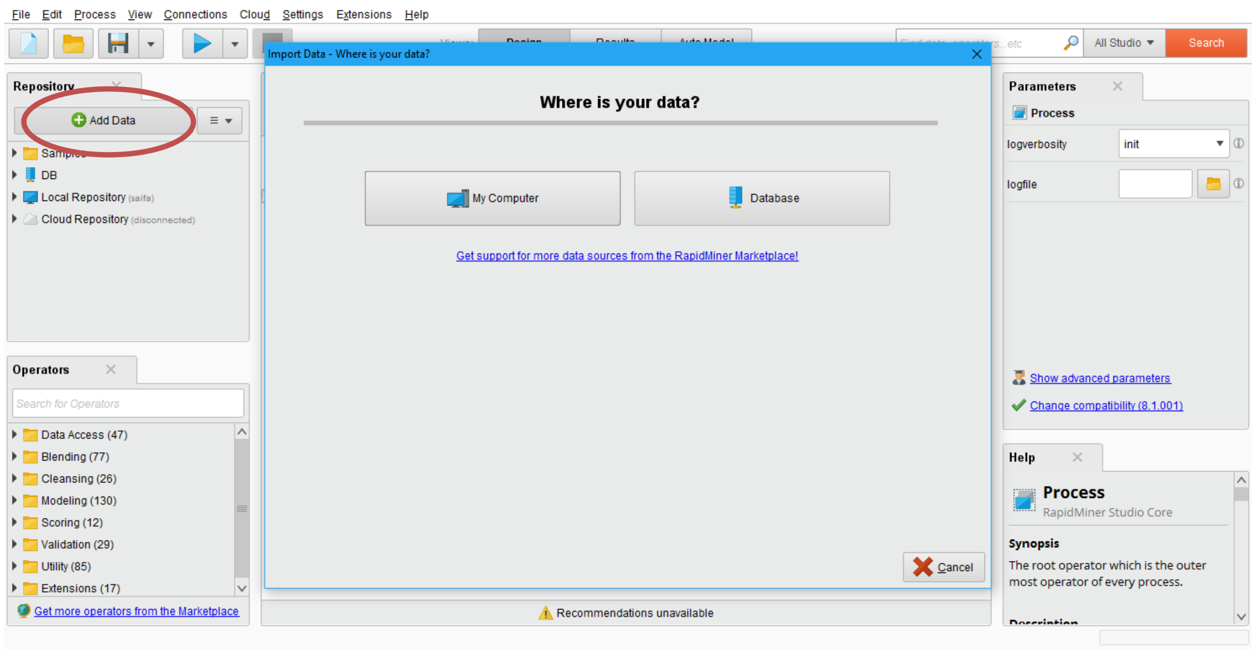
Leverage the Wisdom of Crowds to get operator recommendations based on your process design!

Figure 2. Level 1 Classification Process Diagram

Figure 3. Level 2 Classification Process Diagram

Section 2: Unseen Data Evaluation

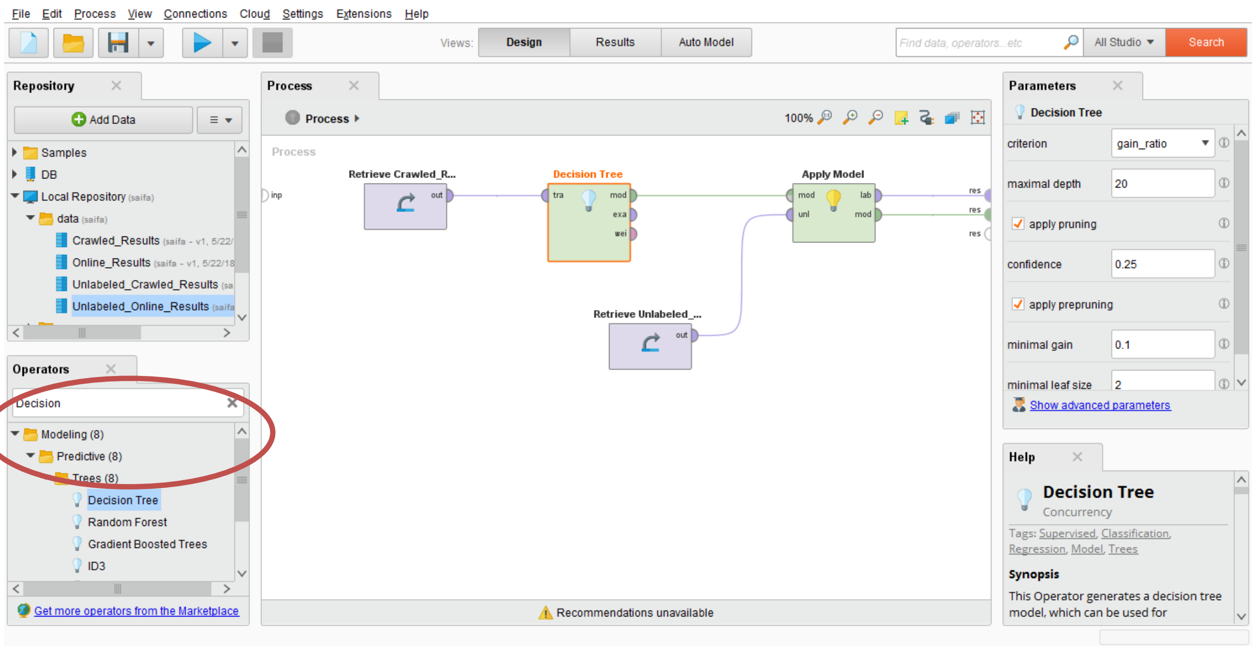
Step 1: Open Blank Process.



The root operator which is the outer most operator of every process.

Step 2: Load the Datasets

Press the Add Data Button to open a new dialogue box for the addition of the Dataset. Navigate to the path where dataset is located.

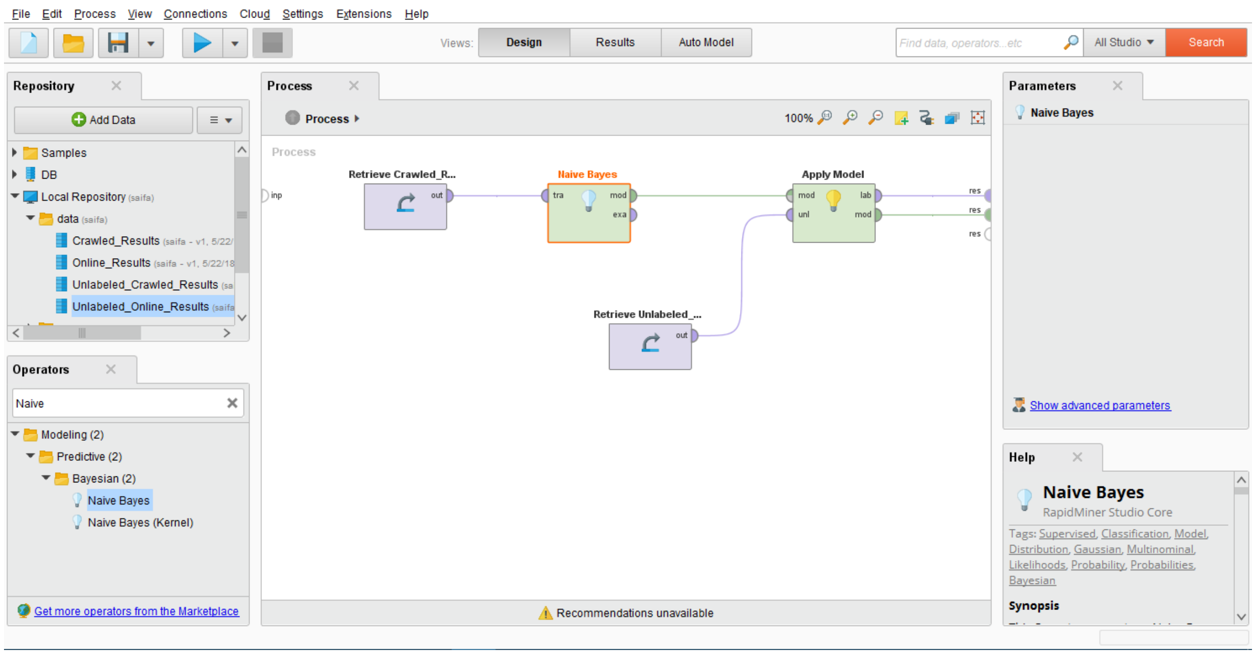


criterion: gain_ratio
maximal depth: 20
apply pruning ✓
confidence: 0.25
apply prepruning ✓
minimal gain: 0.1
minimal leaf size: 2

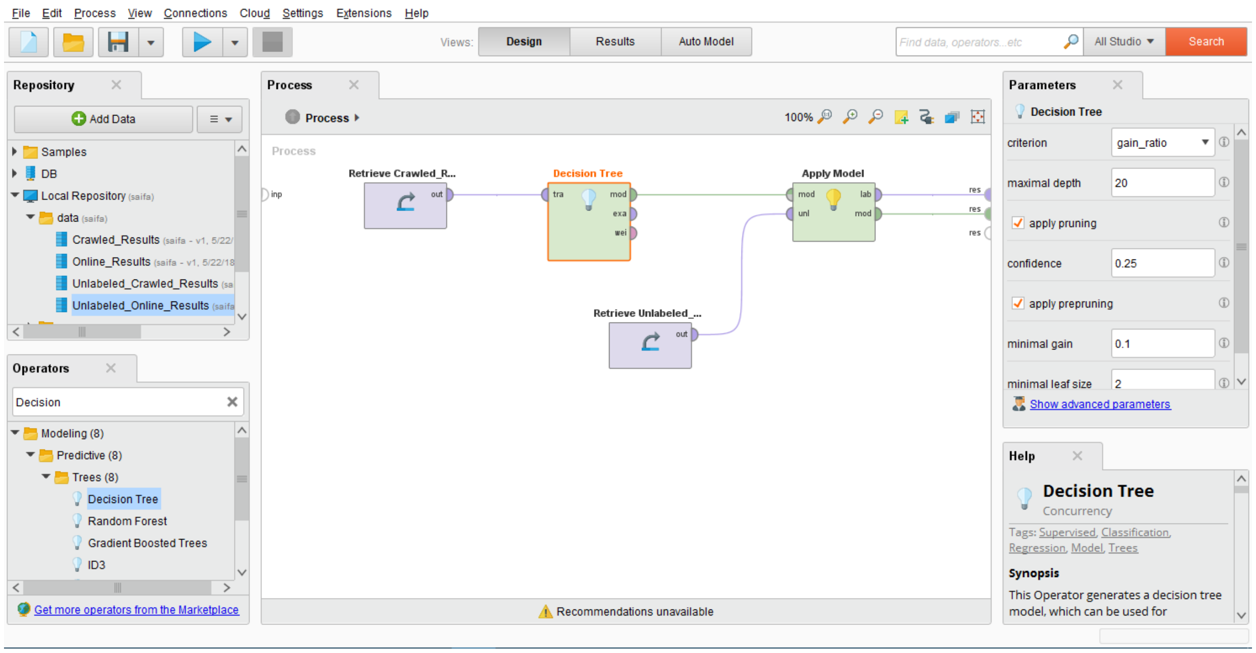
This Operator generates a decision tree model, which can be used for

Step 3: Create the Process

Search for the required operators in the search bar. Join the wires to create the connection


File Edit Process View Connections Cloud Settings Extensions Help
Views: Design Results Auto Model
Find data, operators...etc
All Studio
Search
Repository
Add Data
Samples
DB
Local Repository (saifa)
data (saifa)
Crawled_Results (saifa - v1, 5/22/
Online_Results (saifa - v1, 5/22/18
Unlabeled_Crawled_Results
Unlabeled_Online_Results
Operators
Naive
Modeling (2)
Predictive (2)
Bayesian (2)
Naive Bayes
Naive Bayes (Kernel)
Get more operators from the Marketplace
Process
100%
Retrieve Crawled_R...
Naive Bayes
Apply Model
Retrieve Unlabeled_...
inp
out
tra
mod
exa
unl
lab
res
Recommendations unavailable
Parameters
Naive Bayes
Show advanced parameters
Help
Naive Bayes
RapidMiner Studio Core
Tags: Supervised, Classification, Model, Distribution, Gaussian, Multinominal, Likelihoods, Probability, Probabilities, Bayesian
Synopsis

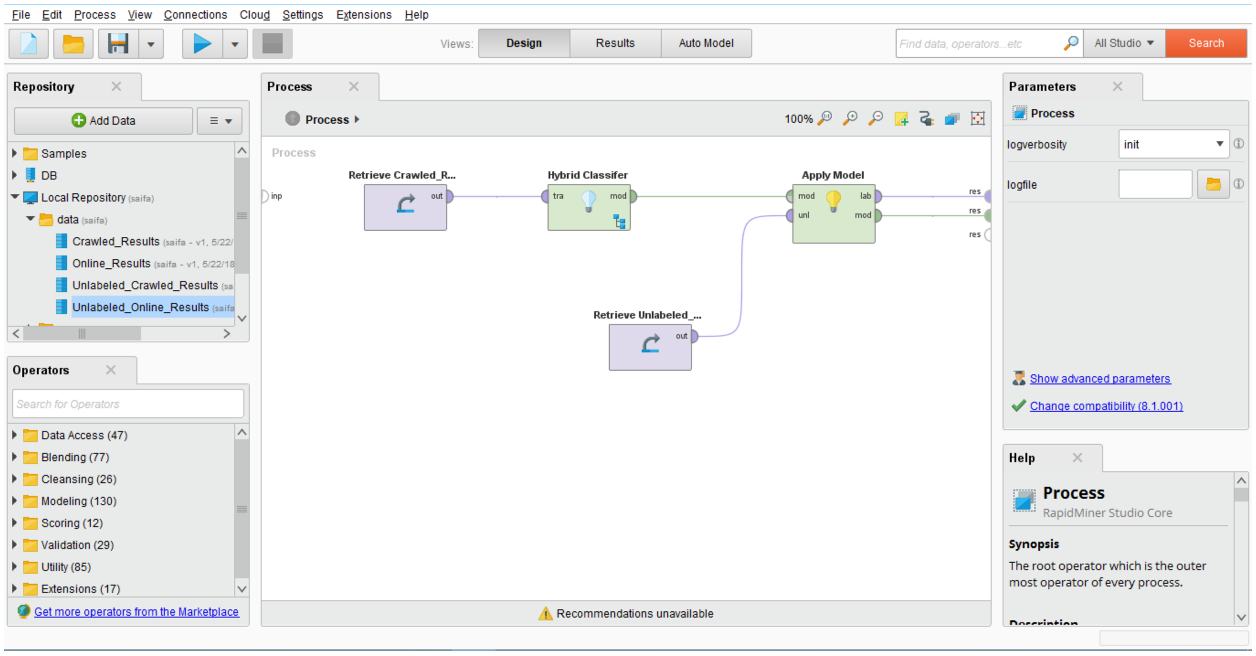
Naïve Bayes Process



criterion	gain_ratio
maximal depth	20
✓ apply pruning	
confidence	0.25
✓ apply prepruning	
minimal gain	0.1
minimal leaf size	2

Decision Tree
Concurrency
Tags: Supervised, Classification, Regression, Model, Trees
Synopsis
This Operator generates a decision tree model, which can be used for

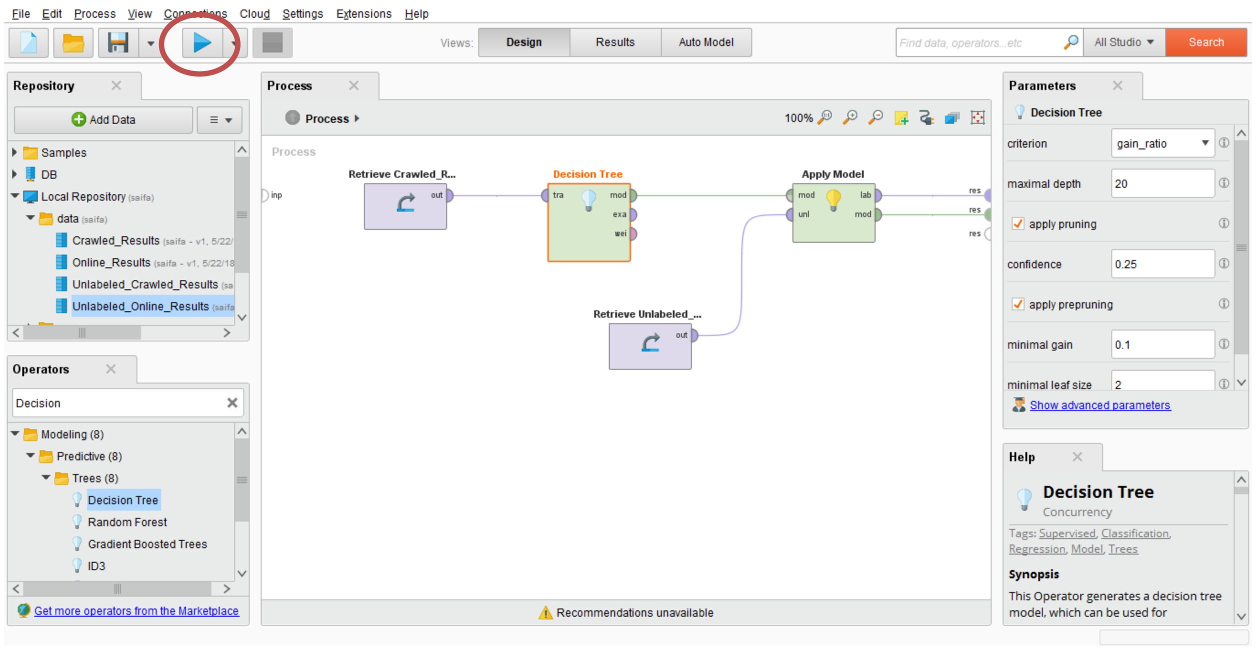
Decision Tree Process



logverbosity	init
logfile	

Process
RapidMiner Studio Core
Synopsis
The root operator which is the outer most operator of every process.

Classifier Process



Decision Tree

criterion	gain_ratio
maximal depth	20
✓ apply pruning	
confidence	0.25
✓ apply prepruning	
minimal gain	0.1
minimal leaf size	2

Decision Tree
Concurrency
Tags: Supervised, Classification, Regression, Model, Trees

Synopsis
This Operator generates a decision tree model, which can be used for