AN ALTERNATIVE SOLUTION FOR HARD DISK DRIVE CLASSIFICATION PROCESS IMPROVEMENT

NATTHAKRITTA RUNGTALAY

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF
MASTER OF ENGINEERING IN DATA STORAGE TECHNOLOGY
INTERNATIONAL COLLEGE
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2016

KMITL-2016-IC-M-005-001

หัวข้อวิทยานิพนธ์	วิธีการทางเลือกเพื่อปรับปรุงกระบวนการการจำแนกฮาร์ดดิสก์ไดร์ฟ
นักศึกษา	นายณัฐกฤตา รุ่งทะเล
รหัสประจำตัว	54600712
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	เทคโนโลยีการบันทึกข้อมูล
พ.ศ.	2559
อาจารย์ที่ปรึกษาวิทยานิพนธ์	ผู้ช่วยศาสตราจารย์ ดร. ชานนท์ วริสาร

บทคัดย่อ

ปัจจุบันนี้กระบวนการปรับปรุงพัฒนาความถูกต้องแม่นยำของกระบวนการทดสอบจำเป็นต้องใช้ความสามารถของมนุษย์ เนื่องด้วยกระบวนการทดสอบมีความซับซ้อนสูงกระบวนการปรับปรุงพัฒนาความถูกต้องแม่นยำของกระบวนการทดสอบจึงเป็นงานที่ยากลำบากและใช้เวลานาน อีกทั้งยังมีค่าใช้จ่ายที่สูงอีกด้วย ดังนั้นในวิทยานิพนธ์นี้จึงได้นำเสนอการใช้เทคนิคด้านปัญญาประดิษฐ์เพื่อใช้เป็นทางเลือกสำหรับการปรับปรุงพัฒนาความถูกต้องแม่นยำของกระบวนการคัดแยกของเสีย

วัตถุประสงค์ของการศึกษาครั้งนี้ คือ การสร้างโมเดลที่มีความถูกต้องแม่นยำสูงและใช้ทรัพยากรระบบต่ำเพื่อนำไปใช้ในการจำแนกฮาร์ดดิสก์ที่ถูกคัดแยกผิด

วิทยานิพนธ์ฉบับนี้ได้นำเสนอการใช้เทคนิคด้านปัญญาประดิษฐ์เพื่อจำแนกฮาร์ดดิสก์ที่ถูกคัดแยกผิด ฮาร์ดดิสก์ที่ถูกคัดแยกผิดนั้นสามารถแบ่งได้เป็นสองประเภท คือ ฮาร์ดดิสก์ที่มีคุณสมบัติไม่ตรงตามความต้องการที่ถูกจำแนกเป็นฮาร์ดดิสก์ที่มีคุณสมบัติตรงตามความต้องการ (false pass) และฮาร์ดดิสก์ที่มีคุณสมบัติตรงตามความต้องการที่ถูกจำแนกเป็นฮาร์ดดิสก์ที่มีคุณสมบัติไม่ตรงตามความต้องการ (false fail) เนื่องด้วยความแตกต่างระหว่างฮาร์ดดิสก์ที่มีคุณสมบัติไม่ตรงตามความต้องการที่ถูกจำแนกเป็นฮาร์ดดิสก์ที่มีคุณสมบัติตรงตามความต้องการ (false pass) และฮาร์ดดิสก์ที่มีคุณสมบัติตรงตามความต้องการที่ถูกจำแนกเป็นฮาร์ดดิสก์ที่มีคุณสมบัติไม่ตรงตามความต้องการ (false fail) ผู้เขียนจึงได้เลือกใช้วิธีจัดแบ่งกลุ่มสองวิธี คือ วิธีการจัดแบ่งกลุ่มโดยใช้ข้อความ (text classification) และวิธีการจัดแบ่งกลุ่มโดยใช้หลักการทางสถิติ (statistical classification) โดยวิธีการจัดแบ่งกลุ่มโดยใช้ข้อความ (text classification) นั้นใช้กับฮาร์ดดิสก์ที่มีคุณสมบัติไม่ตรงตามความต้องการที่ถูกจำแนกเป็นฮาร์ดดิสก์

ที่มีคุณสมบัติตรงตามความต้องการ (false pass) และวิธีการจัดแบ่งกลุ่มโดยใช้หลักการทางสถิติ (statistical classification) ใช้กับฮาร์ดดิสก์ที่มีคุณสมบัติตรงตามความต้องการที่ถูกจำแนกเป็นฮาร์ดดิสก์ที่มีคุณสมบัติไม่ตรงตามความต้องการ (false fail) โดยการศึกษากับการจัดแบ่งกลุ่ม 4 อัลกอริทึม ประกอบด้วย naive Bayes, *k*-Nearest Neighbor (*k*-NN), artificial neural network และ C5.0 โดยที่ naïve Bayes และ *k*-NN อัลกอริทึมประยุกต์ใช้กับวิธีการจัดแบ่งกลุ่มโดยใช้ข้อความ (text classification) และ artificial neural network และ C5.0 อัลกอริทึมประยุกต์ใช้กับวิธีการจัดแบ่งกลุ่มโดยใช้หลักการทางสถิติ (statistical classification) ขั้นตอนสำคัญขั้นตอนแรกคือการประยุกต์ใช้กระบวนการกรองข้อมูลเพื่อลดการซ้ำซ้อนของข้อมูลและลดข้อมูลที่ไม่จำเป็นเหลือไว้เพียงข้อมูลที่มีความหมายต่อการวิเคราะห์ กระบวนการกรองข้อมูลยังช่วยลดพื้นที่ที่ใช้ในการเก็บข้อมูลและยังช่วยประหยัดทรัพยากรระบบอีกด้วย วิธีการแยกส่วนทดสอบหลายกลุ่มทดสอบได้ถูกนำมาใช้เพื่อวัดประสิทธิภาพของแต่ละอัลกอริทึม การทดลองเพื่อศึกษา artificial neural network อัลกอริทึมประกอบไปด้วย 5 การทดลองดังนี้ การทดลองปรับขนาดของกลุ่มข้อมูลเรียนรู้ การทดลองปรับจำนวนชั้นของชั้นซ่อนตัวชั้นที่หนึ่ง การทดลองปรับจำนวนชั้นของชั้นซ่อนตัวชั้นที่สอง การทดลองปรับค่าอัลฟา และการทดลองปรับค่าอัตราการเรียนรู้ การทดลองเพื่อศึกษา C5.0 อัลกอริทึมประกอบไปด้วย 3 การทดลองดังนี้ การทดลองปรับค่าจำนวนหน่วยข้อมูลที่น้อยที่สุดในหนึ่งกิ่ง (minimum records per child branch) การทดลองปรับค่าจำนวนการบูสต์ติง (boosting iteration) การทดลองปรับค่าความรุนแรงของการตัดแต่ง (pruning severity) การทดลองเพื่อศึกษา *k*-NN อัลกอริทึมประกอบไปด้วย 2 การทดลองดังนี้ การทดลองประยุกต์ใช้ *k*-NN อัลกอริทึมโดยไม่มีการใช้กระบวนการกรองข้อมูลร่วมด้วย การทดลองประยุกต์ใช้ *k*-NN อัลกอริทึมโดยมีการใช้กระบวนการกรองข้อมูลร่วมด้วย การทดลองเพื่อศึกษา naïve Bayes อัลกอริทึมประกอบไปด้วย 1 การทดลองคือ การทดลองประยุกต์ใช้ naïve Bayes อัลกอริทึมโดยมีการใช้กระบวนการกรองข้อมูลร่วมด้วย

จากผลการทดลองพบว่าความสามารถในการคัดแยกรูปแบบผลการทดสอบของแบบจำลองที่ได้จาก *k*-Nearest Neighbor มีความสามารถในการคัดแยกฮาร์ดดิสก์ที่มีคุณสมบัติไม่ตรงตามความต้องการที่ถูกจำแนกเป็นฮาร์ดดิสก์ที่มีคุณสมบัติตรงตามความต้องการ (false pass) 100% และแบบจำลองที่ได้จาก C5.0 มีความสามารถในการคัดแยกฮาร์ดดิสก์ที่มีคุณสมบัติตรงตามความต้องการที่ถูกจำแนกเป็นฮาร์ดดิสก์ที่มีคุณสมบัติไม่ตรงตามความต้องการ (false fail) 98.4%

Thesis Title:	An Alternative Solution for Hard Disk Drive Classification Process Improvement
Student:	Natthakritta Rungtalay
Student ID:	54600712
Degree:	Master of Engineering
Program:	Data Storage Technology
Year:	2016
Thesis Advisor:	Assistant Prof. Dr. Chanon Warisarn

ABSTRACT

Currently, the process to improve test classification accuracy usually requires human abilities. This is a very difficult work for a human due to the complexity of a test process unavoidable, such a working model needs time and cost. Since there are some benefits to use machine learning instead of human, an alternative solution to improve pass-fail classification process is proposed.

The objectives of this study were to develop a high accuracy classification model to classify misclassified drives and minimize classification model system resource consumption.

In this thesis, we introduce the use of machine learning to re-classify misclassified drives. In general, the misclassified drives have two categories: false pass (should have failed), and false fail (should have passed). Because of differences between false pass and false fail failure characteristic, we can use two different classification methods: text classification (false pass drives classification) and statistical classification (false fail drives classification). Four classifiers were selected in this study: the naïve Bayes, *k*-nearest

neighbors (*k*-NN), artificial neural network, and C5.0. The naïve Bayes and *k*-NN used for text classification. Whilst artificial neural network and C5.0 used for statistical classification. The first important step is to apply a data filtering process to eliminate redundant information and reduce unnecessary data. It also helps reducing space and system requirements. Multiple split tests method used for evaluating machine learning algorithms. Five experiments were set-up for the artificial neural network classifier: training set size experiment, hidden layer one node experiment, hidden layer two node experiment, alpha value experiment, and the learning rate experiment. Three experiments were set-up for the C5.0 classifier: minimum records per child branch experiment, number of boosting iteration experiment, and pruning severity experiment. Two experiments were set-up for the *k*-NN classifier: *k*-NN classifier without data filtering process experiment and *k*-NN classifier with data filtering process experiment. An experiment was set-up for the naïve Bayes classifier, which is the naïve Bayes classifier with data filtering process experiment.

Our experimental results suggest that it is feasible to determine which classifiers can predict incorrectly classified pass drives and incorrectly classified fail drives. We found that *k*-NN classifier can offer 100% accuracy for incorrectly classified pass drives. For incorrectly classified fail drives, C5.0 classifier can offer 98.4% accuracy.

ACKNOWLEDGEMENT

This thesis would not have been possible without the guidance and the support of several persons who contributed and extended their appreciated assistance in the completion of this research.

First, I would like to sincerely thank The College of Data Storage Innovation King Mongkut's Institute of Technology Ladkrabang and also Seagate Technology (Thailand) for their equipment and financial supports.

I am grateful to National Science and Technology Development Agency (NSTDA) for the scholarship support.

I am utmost gratitude to my advisor Assistant Prof. Dr. Chanon Warisarn, who has given me valuable suggestions and useful advice to overcome all the obstacles in the completion of this research.

Special thanks for my American best friend Erwin Daniel Dunbar, who has given me priceless suggestions about English grammar.

Finally, I am really grateful to my family for all of the love, caring and understanding throughout my life.

Natthakritta Rungtalay

CONTENTS

CONTENTS (Cont.)

CONTENTS (Cont.)

LIST OF FIGURES

LIST OF FIGURES (Cont.)

LIST OF FIGURES (Cont.)

LIST OF TABLE

CHAPTER 1

INTRODUCTION

1.1 Backgrounds and Problem Statement

In each step of the test process, a large amount of data is generated called the testing log. Testing logs are used during the hard disk drive (HDD) test process for recording the testing history of individual drives. Testing logs are typically used for failure analysis to diagnose the failure symptom when drives fail. Failed drives are separated from passing drives by a fail event. Possible results of the drive in test process consist of true pass, true fail, false pass (should have failed), and false fail (should have passed) as shown in Figure 1.1. The ideal process should have only two portions: true pass and true fail.

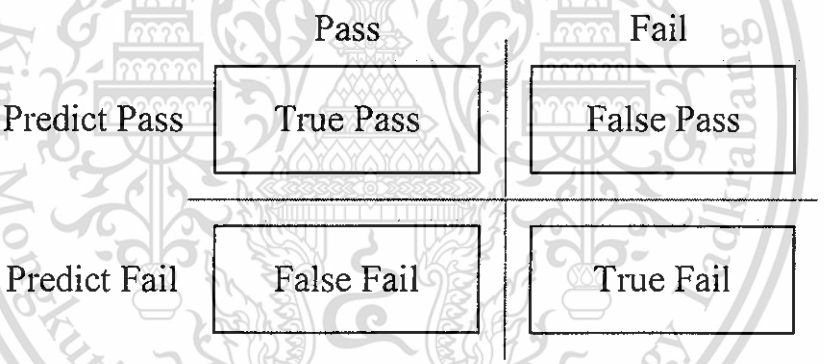

Figure 1.1 Possible results in test process.

The false fail portion is sometimes called over rejection. It unnecessarily increases costs because this portion should have been classified as passing drives instead of waste that requires treatment and disposal costs. Often, the process to reduce false fail drive is to rerun some failed drives to make sure that those drive are true fail drives (repeatability). The low percent repeatability means the fail drive has low chance to repeat fail so the factory has high chance to get good drive. Because of a lot of false fail drive failure symptoms are mixed in the test process, the lowest percent repeatability was selected to study first in this thesis. Currently, the lowest percent repeatability is

when drives fail due to the delta fly height between the outer zone and inner zone being over a failure limit.

The false pass portion is the most serious issue in the testing process. It impacts the reliability and quality of test process. Currently, the factory has an additional screening to capture this kind of failure. However, this screening is costly. Since a lot of drives are tested in a day, it is hard to find false pass and false fail drives which are mixed with millions of true pass and true fail drives. As such, there are clear benefits to more automated and proactive system management. Therefore, this paper proposes a method to establish a prediction model for classifying the incorrectly classified drives. We collected 4,842 drive testing profiles to build a high accuracy prediction model. Within the large amount of data in a testing log, there is some useful information and some not so useful information. A data filtering process [1] used to eliminate redundant information and reduce unnecessary data was an important first step to allow meaningful analysis of log data, which also helped reduce space and system requirements. We filter the log data and process it to be used in classification and prediction algorithms. We applied a number of classification and prediction algorithms such as the naïve Bayes [2-4], C5.0 [5], artificial neural network [6], and *k*-NN algorithm [7] which can classify the false fail drives with 100% accuracy.

1.2 Objective

The objectives of this study were to develop a high accuracy classification model to classify misclassified drives and minimize classification model system resource consumption. To achieve this goal, the specific objectives are as follows:

a) To study possibility to apply machine learning based model in the manufacturing process.
b) To study how text classification work.
c) To study how statistical classification work.
d) To study how *k*-NN and the naïve Bayes classifiers classify in text classification.
e) To study how artificial neural network and C5.0 classifiers classify in statistical classification.

f) To study how to develop a high accuracy classification model.

g) To study how to minimize classification model resource consumption.

1.3 Scope of Work

The scopes of this thesis are described below.

a) Use one HDD product model to study.

b) All failure samples collected from a manufacturing database to simulate with computer programing.

c) Four classification algorithms were selected to study: neural network, C5.0, the naïve Bayes, and *k*-NN.

d) Use Clementine 12.0 application to run neural network and C5.0 classifiers.

e) Use RapidMiner 5 application to run the naïve Bayes classifiers.

f) Use self-writing application developed based on the python language to run k-NN classifier.

g) The lowest percent repeatability was selected to study. Currently, the lowest percent repeatability is when drives fail due to the delta fly height between the outer zone and inner zone being over a failure limit.

1.4 Expected Benefits

a) Reduce manufacturing classification process cost.

b) Develop an automatic classification tool.

1.5 Research Outline

This thesis is organized into five chapters as follow:

a) Introduction: presents the background, objectives, scope of work and benefit of the thesis.

b) Theory: explains the related theory. The first topic is the hard disk drive test process overview. The second topic is adaptive fly height test state characteristic. The third topic is fly height measurement in hard disk drive manufacturing which has two sub-topics: contact detection, and fly height measurement results. The

fourth topic is data mining. The fifth topic is the naïve Bayes classifier for text classification which has two sub-topics: advantages of the naïve Bayes classifier, and disadvantages of the naïve Bayes classifier. The sixth topic is *k*-nearest neighbors classifier for text classification which has two sub-topics: advantages of *k*-nearest neighbors classifier, and disadvantages of *k*-nearest neighbors classifier. The seventh topic is artificial neural network classifier which has eight sub-topics: the perceptron concept, single layer perceptron networks, multilayer perceptron networks, back-propagation, gradient descent method, back-propagation rules, advantage of artificial neural network classifier, and disadvantages of artificial neural network classifier. The eighth topic is C5.0 classifier which has two sub-topics: advantages of C5.0 classifier, and disadvantages of C5.0 classifier. The ninth topic is literature review which has three sub-topics: customer failure modes prediction for hard disk drive using neural network rank-level fusion, hard disk drive failure mode prediction from SMART attributes using data mining method, and study of neural network for fly height failure pattern classification in hard disk drive.

c) Research methodology: present the description of the research methodology. The first topic is the measurement system which has two sub-topics: repeatability in measurement system, and novel system for recovering failed drive. The second topic is data and sample collection. The third topic is data analysis which has two sub-topics: appearance of false fail drives, and appearance of false pass drives. The fourth topic is data pre-processing which has two sub-topics: data pre-processing for false pass drives, and data pre-processing for false fail drives. The fifth topic is classification process which has two sub-topics: classification system, and classification evaluation.

d) Experimental and results: presents the evaluation and performance comparison of false pass and false fail classification, which is associated to classifier and settings. The first topic is artificial neural network classifier experiments which have five experiments: training set size experiment, hidden layer one node experiment, hidden layer two node experiment, alpha value experiment, and the learning rate

experiment. The second topic is C5.0 classifier Experiments which have three experiments: minimum records per child branch experiment, number of boosting iteration experiment, and pruning severity experiment. The third topic is *k*-NN classifier experiments which have two experiments: *k*-NN classifier without data filtering process experiment, and *k*-NN classifier with data filtering process experiment. The fourth topic is naïve Bayes classifier experiments

e) Conclusions: presents the conclusions and recommendations for the model as suitable to classify false fail drives and false pass drive in HDD manufacturing.

CHAPTER 2

THEORY

2.1 The Hard Disk Drive Test Process Overview

In the HDD industry, every hard disk drive is tested against customer specification. Test process is a process for sorting out hard disk drives that do not meet specifications. The sequence starts with the PWA operation which installs a PCBA (Printed Circuit Board Assembly) onto the HDA, and then the fully assembled component hard disk drives are brought to test in the tester, which for examples consists of instant AFH (Adaptive Fly Height) and VBAR (Variable Bit Aspect Ratio) for individual head fly height measuring and drive capacity measuring respectively. Finally, all of passer drives are pack and ready to ship to customer as shown in Figure 2.1.

Figure 2.1 Simplified test process flow.

2.2 Adaptive Fly Height Test State Characteristic

In the test process, "Adaptive Fly Height" is one of the test process sequences which can measure fly height and adjust fly height to appropriate fly height. This sequence can reject the drive with some criteria. For example, if the delta fly height between the outer zone and the inner zone is over a certain specification. A Fly height profile of a drive failing such a criteria is shown Figure 2.2.

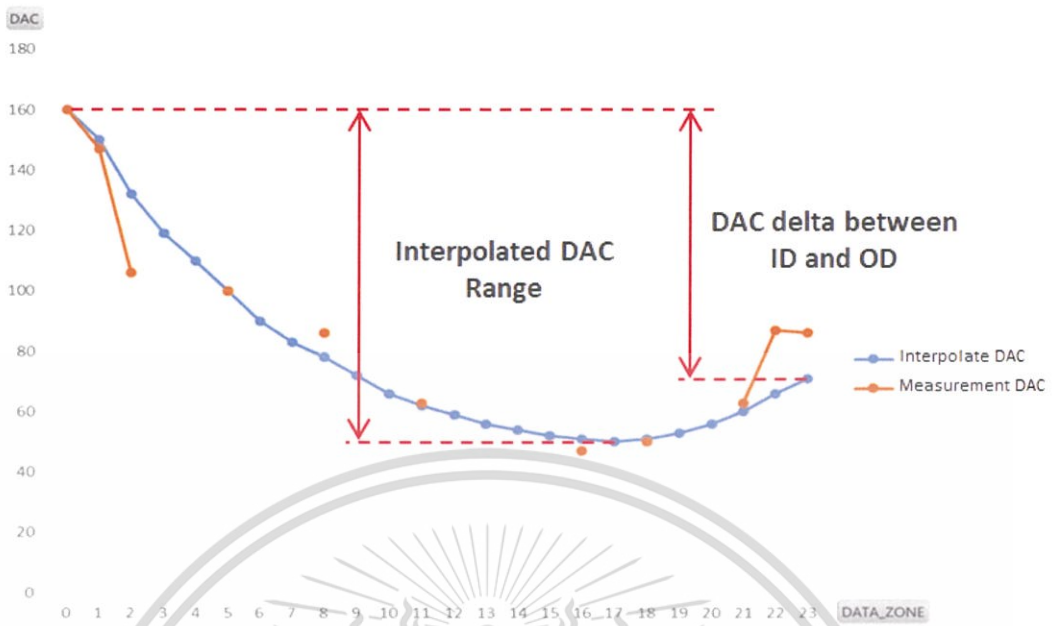

Figure 2.2 Fly height profile of failed drive.

2.3 Fly Height Measurement in Hard Disk Drive Manufacturing

Fly height (FH) measurement in the hard disk drive (HDD) manufacturing process uses thermal fly-height control (TFC) technology to measure the spacing between the recording head and the disk by applying power to the heater element of the recording head until it contacts the disk. Figure 2.3 shows elements of the recording head (the reader and writer) get closer to the disc while applying power to the heater element. Figure 2.4 shows that without applying power to the heater element, the reader and writer fly further away from the disc.

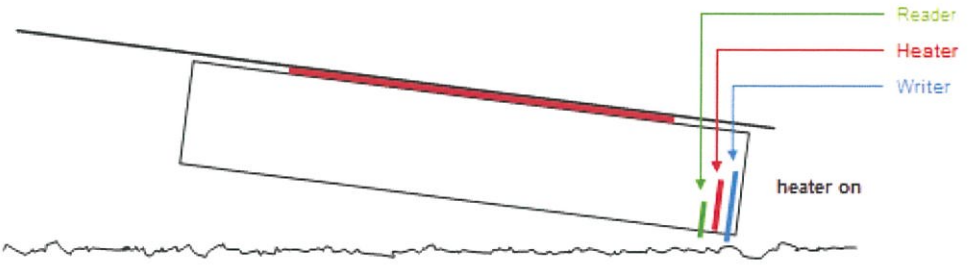

Figure 2.3 Elements of the recording head when applying power to the heater element.

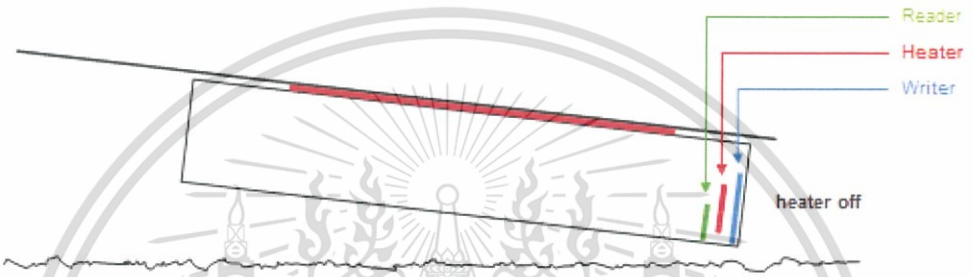

Figure 2.4 Elements of the recording head without applying power to the heater element.

The objective of FH measurement is to determine the most suitable heater power in each data zone to perform read and write operations. The clearance target requirement is less than three nanometers from the disk [8], so FH measurement is an important drive testing function, as one of the critical methods for increasing areal density capacity (ADC) is FH reduction. Figure 2.5 presents a schematic diagram of the head–disk interface [9].

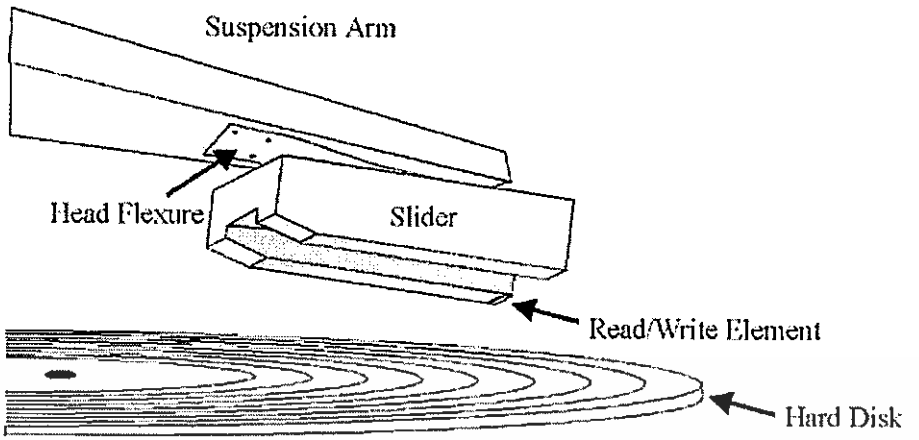

Figure 2.5 Schematic diagram of the head-disk interface. [9]

2.3.1 Contact Detection

FH measurement requires specific head and data zone testing. Several attributes and designs, the air bearing surface (ABS) pattern, the mechanical assembly, including the head gimbal assembly (HGA)/suspension and the lubricant type on the disk all contribute to different FH characteristics.

A widely used technique for contact detection on the drive level is applying the servo system with a position error signal (PES). The servo system controls the position of the recording head, and moves the head stack assembly (HSA) from the current to another desired track. The embedded servo shown in Figure 2.6, has a magnetic pattern (servo pattern) to generate the signal when the reader passes over it. Each of the servo sectors are written by a specific pattern of magnetization, with a unique number for each track. Thus, the embedded servo information identifies which track the recording head is on. A servo sector consists of a DC-gap field, automatic gain control (AGC) field, servo timing mark (STM) field, grey coded track ID field, and PES burst pattern field.

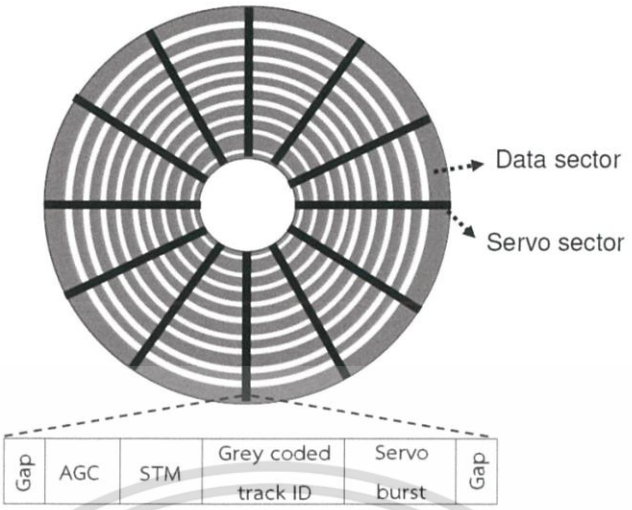

Figure 2.6 Embedded servo on a disk. [8]

The PES is generated using utilization from the grey coded track ID field and PES burst pattern field. The grey coded track ID field contains information of the track number, while the PES burst pattern is used to measure the off-track position of the reader from the track center. Every time that the reader come out of center track, the PES signal will rise and feed back to servo controller to control the reader back to the center track again. as shown in Figure 2.7.

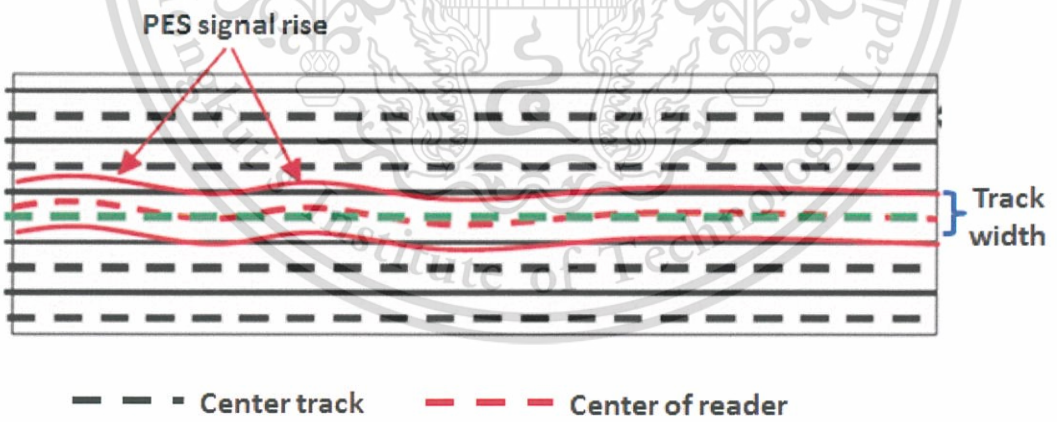

Figure 2.7 PES signal rising process.

A simple example for servo burst patterns is shown in Figure 2.8. For a setting using A/B burst patterns, when the reader passes over the burst pattern, a read back signal will be generated [10]. For example, when the reader stays on position number 0

(desired track), the amplitude difference between A burst and B burst equals zero. On the other hand, at the center of position number 1, the read back amplitude from B burst is a negative maximum value. The read back amplitude is proportional to the length of the part of reader when it flies over the burst.

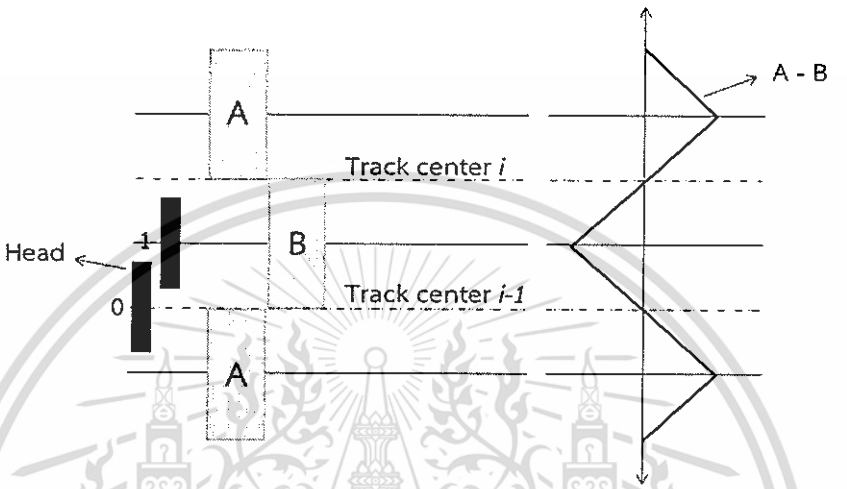

Figure 2.8 Servo patterns with A/B burst to generate off-track error. [8]

PES is computed by the ratio of the difference between the signal amplitude coming from A and B bursts as shown equation as follows:

$$PES = \frac{A - B}{A + B}, \tag{2.1}$$

where A and B represent the track average amplitude (TAA) of the A and B bursts, respectively [11].

The FH measurement utilizes PES to detect a contact incident. When applying heater power, the recording head will be protruded. Every step of the increasing voltage bias to the heater is monitored by PES, using the fast Fourier transform (FFT) algorithm. When the recording head touches the lubricant of the disk, this produces a force between the head and the disk that generates an off-track error. The contact force is

perpendicular to slider. This sideways force pushes the head off-track as shown in Figure 2.9.

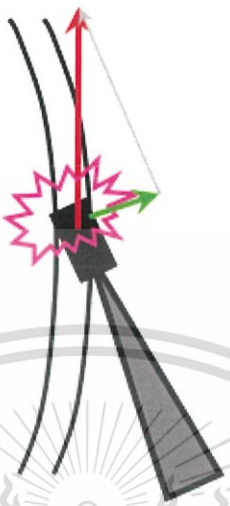

Figure 2.9 The contact force diagram.

The contact detect algorithm is called dPES (delta position error signal). In Figure 2.10, a significant outlier from the baseline signal that will be considered to be the contact signal.

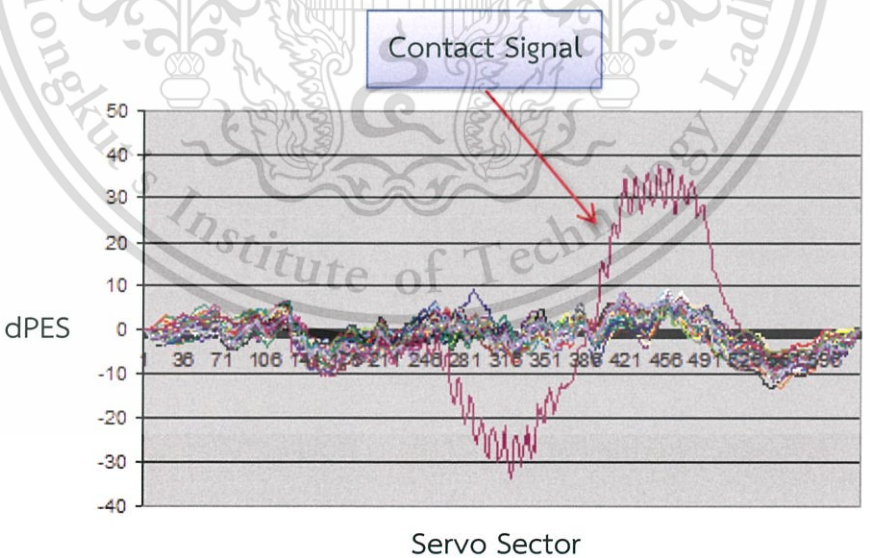

Figure 2.10 The contact signal.

The change in PES is detected by the FFT algorithm, and identifies the passive FH distance in digital to analog converter (DAC) units between the recording head and the disk. For the DAC, it is an 8-bit value that the preamp uses to control the voltage of the heater. It can also convert the power to milliwatts, using the formula from the preamp vendor's datasheet. However, this research considers only data pattern, so DAC from log-file will be not transformed to milliwatt units, because DAC is directly proportional.

2.3.2 Fly Height Measurement Results

In this research, the value of contact power represents the FH distance. Figure 2.11 shows the FH distance of a recording head, where the y axis is the contact heater power value, and the x axis is the data zone of a disk in order of the outer diameter towards the inner diameter, composed of ten data points for each read/write operation. The FH distance for the write operation is always lower than the read operation, because the thermal power from the writer element during the write operation causes more protrusion.

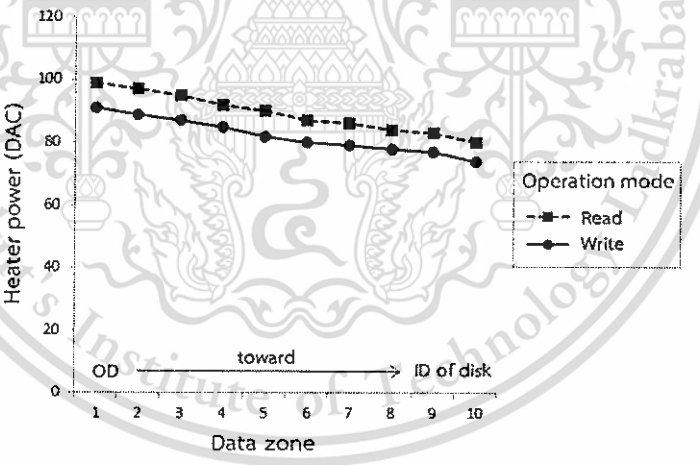

Figure 2.11 FH distance of a recording head.

After completing FH measurements, the head-disk spacing change at various levels of heater power was computed from the basics of the Wallace spacing loss equation: [12, 13], defined as follows:

$$\Delta FH = \frac{\lambda}{2\pi}\ln(\frac{A_1}{A_2}), \tag{2.2}$$

where ΔFH is the difference in FH between the heater power and the reference, λ is the written wavelength, $\ln$ is the natural logarithm, A_1 is the reference read-back signal, A_2 is the amplitude of the heater power, and A_1/A_2 is the amplitude ratio of the read-back signal. Therefore, each step of applied heater power is proportional to the change in head-disk spacing. The result of the relation between ΔFH from the reference (delta FH = 0) on the y axis, versus the various levels of heater power on the x axis is illustrated in Figure 2.12.

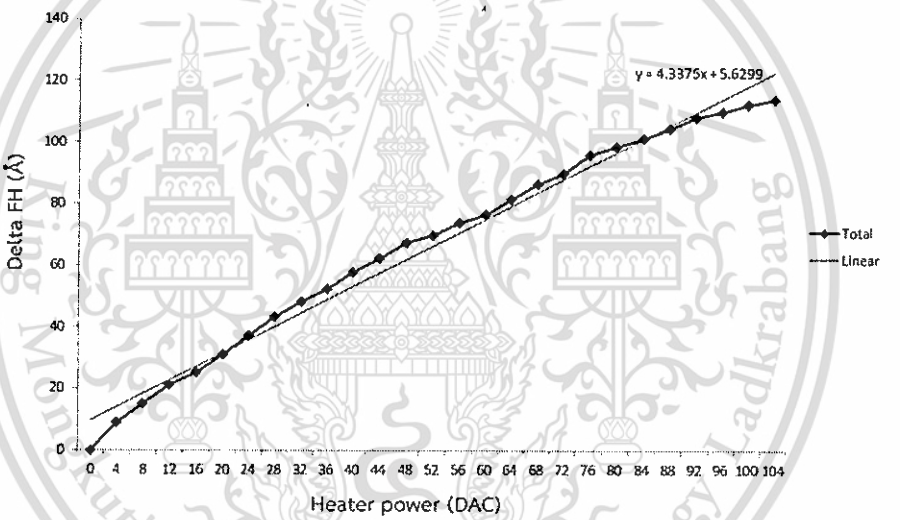

Figure 2.12 Delta FH at various levels of heater power. [8]

Finally, the contact distance is calculated by computing the active FH (working) distance from the formula in equation as follows:

$$AFD = CTD - \mathrm{TGD}, \tag{2.3}$$

where AFD is active FH distance, CTD is contact distance, and TGD is FH target distance. The FH target distance is the appropriate spacing from the disk to the recording

head to perform read and write operations. The target distance depends on product design. For example, 20 angstroms from the disk will achieve the areal density to reach the product's requirement. Figure 2.13 shows a schematic diagram of the contact distance, the target read FH distance, and the target write FH distance.

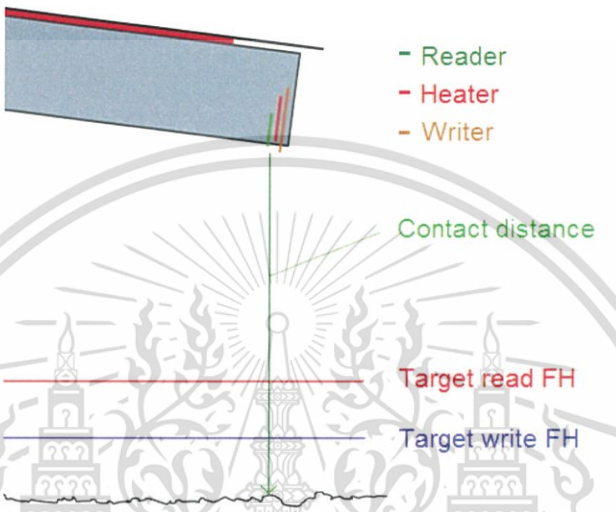

Figure 2.13 Schematic diagram of the contact distance, the target read FH distance, and the target write FH distance.

2.4 Data Mining

Data mining [14, 15] is the process of discovering interesting knowledge, such as patterns, association, changes, anomalies and significant structure, from large amounts of data stored in databases, data warehouses, or other information repositories. Due to the wide availability of huge amounts of data in electronic forms, and the imminent need for turning such data into useful information and knowledge for broad application including market analysis, business management, and decision support, data mining has achieved a great deal of attention in the information industry in recent years [16].

Data mining has been popularly treated as a synonym of knowledge discovery in databases, although some researchers view data mining as an essential step of knowledge discovery. In general, a knowledge discovery process consists of an iterative sequence of the following steps:

- Data cleaning, which handles noisy, erroneous, missing, or irrelevant data.
- Data integration, where multiple, heterogeneous data sources may be integrated into one.
- Data selection, where data relevant to the analysis task are retrieved from the database.
- Data transformation, where data are transformed or consolidated into forms appropriate for mining by performing summary of aggregation operations.
- Data mining, which is an essential process where intelligent methods are applied in order to extract data patterns.
- Pattern evaluation, which is to identify the truly interesting patterns representing knowledge based on some interestingness measure.
- Knowledge presentation, where visualization and knowledge representation techniques are used to present the mined knowledge to the user.

In this research, a data mining system is applied to accomplish the Classification task, which is one of the major tasks of data mining.

In general, data mining tasks can be classified into two categories: descriptive data mining and predictive data mining. The former describes the dataset in a concise and summary manner and presents interesting general properties of the data; whereas the latter constructs one or a set of models, performs inference on the available set of data, and attempts to predict the behavior of new data sets.

A data mining system may accomplish one or more of the following data mining tasks.

1. Class description. A class description provides a concise and succinct summarization of a collection of data is called class characterization; whereas the comparison between two or more collections of data is called class comparison or discrimination. Class description should cover not only its summary properties, such as count, sum, and average, but also its properties on data dispersion, such as variance, quartiles, etc. For example, a class description can be used to compare European versus Asian sales of a company, identify the important factors which discriminate the two classes, and present a summarized overview.

2. Association. Association is the discovery of association relationships or correlations among a set of items. They are often expressed in the rule form showing attribute-value conditions that occur frequently together in a given set of data. An association rule in the form of X => Y is interpreted as "database tuples that satisfy X are likely to satisfy Y". Association analysis is widely used in transaction data analysis for directed marketing, catalog design, and other business decision making processes. Substantial research has been performed recently on association analysis with efficient algorithms proposed, including the level-wise Apriori search, mining multiple-level, multi-dimensional associations, mining associations for numerical, and interval data, meta-pattern directed or constraint-based mining, and mining correlations.

3. Classification. Classification analyzes a set of training data (i.e., a set of object whose class label is known) and constructs a model for each class based on the features in the data. A decision tree or a set of classification rules is generated by such a classification process, which can be used for better understanding of each class in the database and for classification of future data. For example, one may classify diseases and help predict a kind of disease based on symptoms of patients. There have been many classification methods developed in the fields of machine learning, statistics, databases, neural networks, rough sets, and others. Classification has been used in customer segmentation, business modeling, and credit analysis.

4. Prediction. This mining function predicts the possible values of some missing data or the value distribution of certain attributes relevant to the attribute of interest (e.g., by some statistical analysis) and predicting the value distribution based on the set of data similar to the selected object(s). For example, an employee's potential salary can be predicted based on the salary distribution of similar employees in the company. Usually, regression analysis, generalized linear model, correlation analysis and decision trees are useful tools in quality prediction. Genetic algorithms and neural network models are also popularly used in prediction.

5. Clustering. Clustering analysis is used to identify clusters embedded in the data, where a cluster is a collection of data objects that are "similar" to one another. Similarity can be expressed by distance functions, specified by users or experts. A good

clustering method produces high quality clusters to ensure that the inter-cluster similarity is low and the intra-cluster similarity is high. For example, one may cluster the houses in an area according to their house category, floor area, and geographical locations. Data mining research has been focused on high quality and scalable clustering methods for large databases and multidimensional data warehouses.

6. Time-series analysis. Time-series analysis is to analyze large set of time-series data to find certain regularities and interesting characteristics, including search for similar sequence or subsequences, and mining sequential patterns, periodicities, trends and deviations. For example, one may predict the trend of the stock values for a company based on its stock history, business situation, competitors' performance and current market.

2.5 The Naïve Bayes Classifier for Text Classification

The naïve Bayes classifier [17, 18] algorithm: Let $C = (c_1, \dots, c_m)$ be m document classes. Given a new unlabeled document D and its corresponding word-list $\bar{W} = (w_1, \dots, w_{d'})$ (defined in the same way as the word-list for the training set), the naïve Bayes approach assigns D to a class C_{NB}^* as follows [17]:

$$C_{NB}^* = \arg\max_{c_j \in C} P(c_j) \prod_{i=1}^{d'} P(W_i|C_j), \tag{2.4}$$

where $P(C_j)$ is the a priori probability of class C_j and $P(W_i|C_j)$ is the conditional probability of word W_i given class C_j. The underlying assumption of the Naïve Bayes approach is that for a given class C_j the probabilities of words occurring in a document are independent of each other. When the size of the training set is small, the relative frequency estimates of probabilities, $P(W_i|C_j)$, will not be reasonable; if a word never appears in the given training data, its relative frequency estimate will be zero. Instead, we applied the Laplace law of succession to estimate $P(W_i|C_j)$. The estimate of the probability $P(W_i|C_j)$ is given as:

$$P(W_i|C_j) = \frac{n_{ij} + 1}{n_j + k_j}, \quad (2.5)$$

where n_j is the total number of words in class c_j, n_{ij} is the number of occurrences of word w_i in class c_j and k_j is the vocabulary size of class c_j. This is the result of the Bayesian estimation with a uniform prior assumption, i.e. probabilities of the occurrence of words appearing in class c_j are equally likely.

2.5.1 Advantages of the Naïve Bayes Classifier

The naïve Bayes is potentially good at serving as a document classification model due to its simplicity. Many researchers have found that the naïve Bayes achieves very good performance in their experiments [2, 3, and 19]. Moreover, there are many advantages of the naïve Bayes as described follows.

a) Conceptually very easy to understand.
b) Simplicity and efficiency.
c) Requires a small amount of training data.
d) Fast to train (single scan). Fast to classify.
e) Not sensitive to irrelevant features.
f) Handles real and discrete data.
g) The method is well defined in cases of missing attributes: training or test examples where some values are not observed.

2.5.2 Disadvantages of the Naïve Bayes Classifier

However, for all Naïve Bayes advantages and positive aspects, the naïve Bayes also has its disadvantage as describe follows.

a) Assumption: class conditional independency, therefore loss of accuracy.
b) Practically, dependencies exist among variables.
c) Dependencies among variables cannot be modeled by the naïve Bayes classifier.

2.6 *k*-Nearest Neighbors Classifier for Text Classification

The basic concept of *k*-nearest neighbors (*k*-NN) classifier is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbors. For *k*-NN algorithm [7], the training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples.

Figure 2.14 show an example of *k*-NN classification. The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles. If k = 3 (solid line circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle. If k = 5 (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).

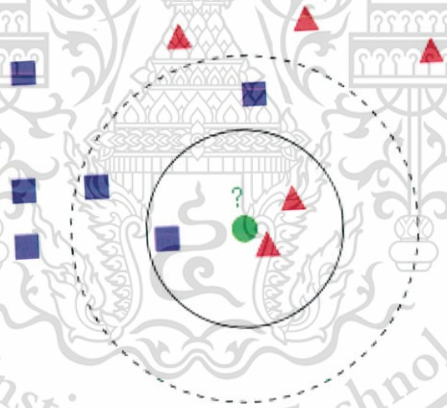

Figure 2.14 Example of *k*-NN classification. [20]

In the classification phase, k is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point. In this thesis, we use the TF-IDF (TF is the term frequency in a document and IDF is the inverse document frequency) weighting scheme and use the cosine similarity [17, 21] instead of Euclidean distance to measure the similarity of the two documents. Given two documents D_1 and D_2, their corresponding weighted feature vectors are $T_1 = (t_{1i}\delta_i 1)_{i=1}^{d}$ and $T_2 = (t_{2i}\delta_i 2)_{i=1}^{d}$,

where δ_{ki} is the weight of word w_i in document k (TF-IDF). The similarity between D_1 and D_2 is then defined as:

$$S(D_1, D_2) = \frac{T_1^T T_2}{\|T_1\| \|T_2\|}, \tag{2.6}$$

where $\|.\|$ denotes the norm of the vector. To save system resource we use k = 1. If k = 1, then the object is simply assigned to the class of its single nearest neighbors. Although, *k*-NN is easy to apply, it is computationally expensive and memory intensive.

2.6.1 Advantages of *k*-Nearest Neighbors Classifier

k-NN is one of the most popular algorithms for text classification [22, 23]. Many researchers have found that the *k*-NN algorithm achieves very good performance in their experiments [24, 25]. There are many advantages be described in detail as follows.

a) Simple and powerful. No need for tuning complex parameters to build a model.
b) Simple to implement.
c) Flexible to feature/distance choices.
d) Naturally handles multi-class cases.
e) Can do well in practice with enough representative data.
f) No training involved ("lazy"). New training examples can be added easily.
g) Complex concepts can be learned by local approximation using simple procedures.

2.6.2 Disadvantages of *k*-Nearest Neighbors Classifier

However, for all *k*-NN advantages and positive aspects, the *k*-NN also has its disadvantage as describe follows.

a) It is computationally expensive to find the *k* nearest neighbors when the dataset is very large.
b) *k*-NN requires a lot of memory.
c) Must know we have a meaningful distance function.

d) The model can not be interpreted (there is no description of the learned concepts).

2.7 Artificial Neural Network Classifier

The neural networks concept has inspiration from the complexity of the human brain consisting of multiple sets of neurons [26, 27]. The initial idea of neural networks was developed as a model for neurons. The first applications of ANNs did not present good results and showed several limitations (such as the treatment of linear correlated data). However, the initial perceptron architecture (a single-layer neural network) was developed into multilayer networks. The introduction of nonlinearity between input and output data and this new architecture of perceptrons yielded positive improvements in ANN results. Moreover, the learning algorithm called "back-propagation" helped the ANN popularization.

In general, ANN techniques [26] are a family of mathematical models that are based on the human brain functioning. All ANN methodologies share the concept of "neurons" (also called "hidden units") in their architecture. Each neuron represents a synapse as its biological counterpart. Therefore, each hidden unity consists of activation functions that control the propagation of one neuron signal to the next. A hidden unit is adjusted by a regression equation that processes the input information into a non-linear output data. Therefore, if more than one neuron is used to compose an ANN, non-linear correlations can be developed. Because of the non-linearity between input and output, we compare the hidden unities of ANNs to a "black box". Figure 2.15 shows a comparison between a human neuron and an ANN neuron [27].

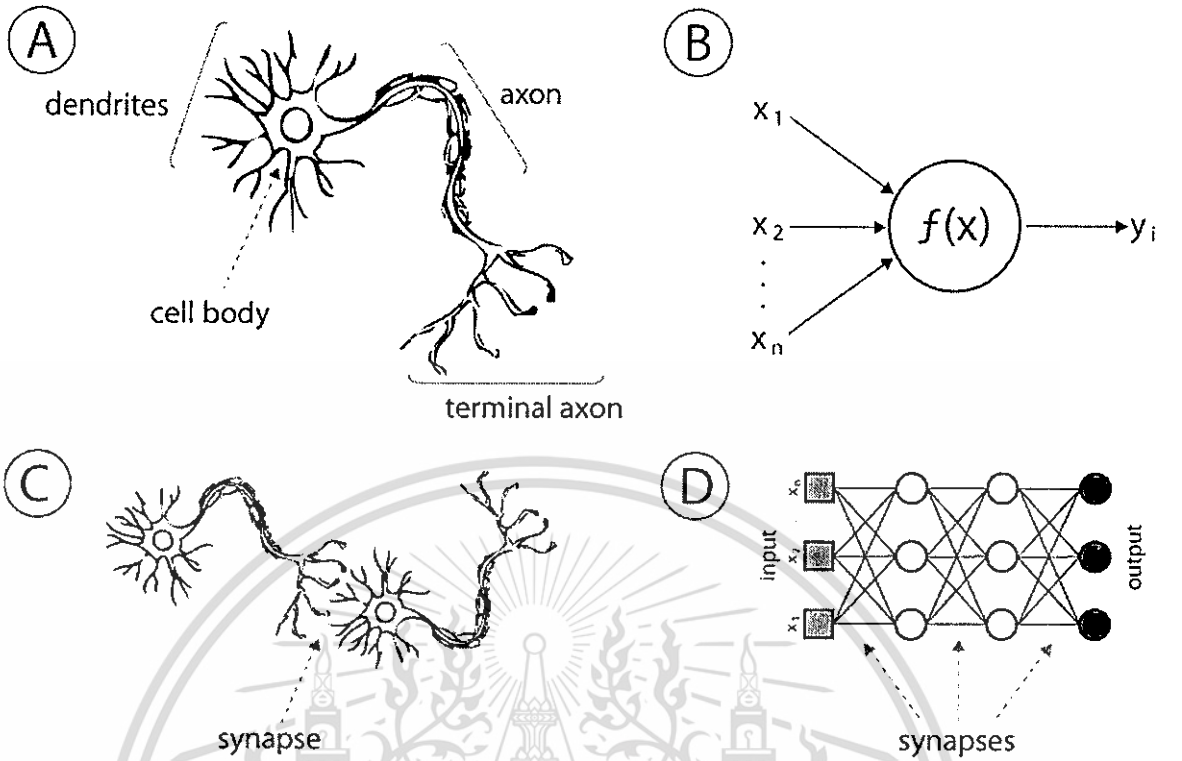

Figure 2.15 (A) human neuron; (B) artificial neuron or hidden unity; (C) biological synapse; (D) neural network synapses. [27]

The general purpose of ANN techniques is based on stimulus–response activation functions that accept some input and yield some output. The difference between the neurons of distinct artificial neural network consists in the nature of activation function of each neuron. There are several standard activation functions used to compose ANNs: the threshold function, linear function, and sigmoid function. However, the sigmoid function is the most popular for ANN techniques. Figure 2.16 shows a graph of the sigmoid activation function $f(x) = 1/1 + e^{-x}$ [28].

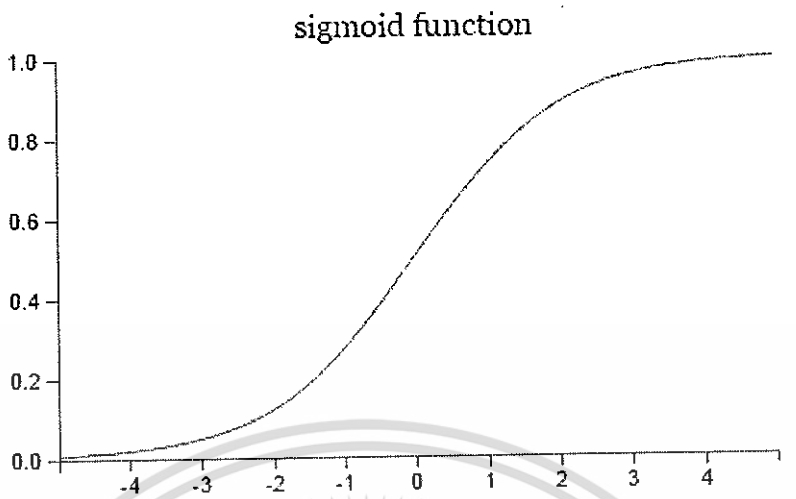

Figure 2.16 Graph of the sigmoid activation function $f(x) = 1/1 + e^{-x}$. [28]

Different ANN techniques can be classified based on their architecture or neuron connection pattern. The feed-forward networks are composed by unidirectional connections between network layers. In other words, there is a connection flow from the input to output direction.

Each ANN architecture has an intrinsic behavior. Therefore, the neural networks can be classified according to their connections pattern, the number of hidden unities, the nature of activation functions and the learning algorithm. There are an extensive number of ANN types and Figure 2.17 exemplifies the general classification of neural networks showing the most common ANN techniques.

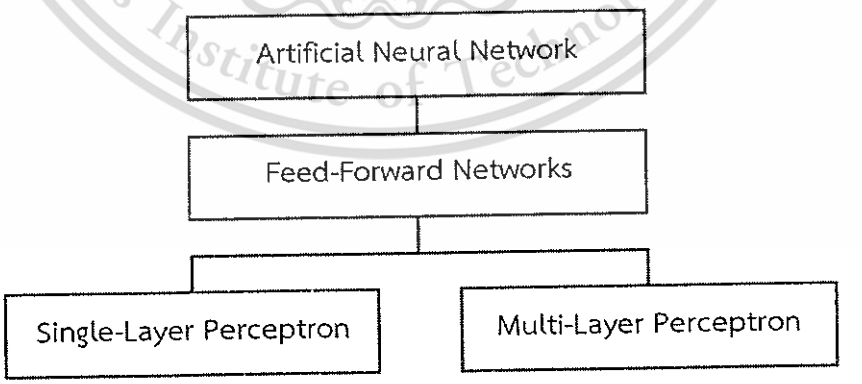

Figure 2.17 The most common neural networks.

According to the previous brief explanation, ANN techniques can be classified based on some features. The next topics explain the most common types of ANN.

2.7.1 The Perceptron Concept

The perceptron is a simple algorithm for binary classification where the weights are adjusted in the direction of each misclassified example [29]. It is simplest neural network possible: a computational model of a single neuron [30]. A perceptron consists of one or more inputs, a processor, and a single output as shown in Figure 2.18.

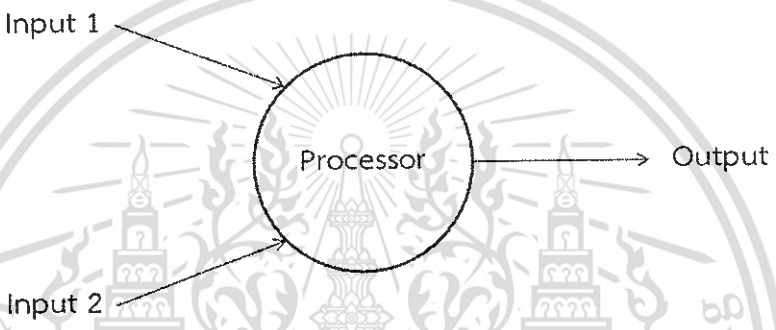

Figure 2.18 The perceptron.

Perceptron follows the "feed-forward" model, meaning inputs are sent into the neuron, are processed, and result in an output. In the diagram above, this means the network (one neuron) reads from left to right: inputs come in, output goes out.

2.7.2 Single Layer Perceptron Networks

Single layer perceptron networks can be presented by the model show in Figure 2.19. Each output consists of the summation of each input multiplied by their weight [31].

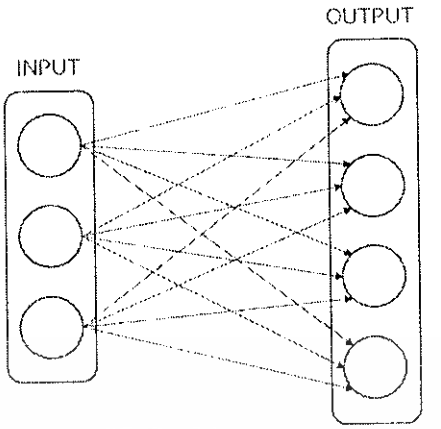

Figure 2.19 Single layer perceptron chart.

The output from model in Figure 2.19 can also be described in Equation as follows.

$$a_j = g\left(\sum_{i=1}^{n}\left(w_{ij}a_i\right)\right), \tag{2.7}$$

where w_{ij} is the connection weight of branch(i,j), a_i is the input data of node i, and g is the activation function.

The number of input nodes depends on the number of input data components and the activation function can be a threshold function (2.8) or sigmoid function (2.9). If the output is a discrete output: 'yes' or 'no', the output function is the threshold function. On the other hand, if the output is a continuous output, the output function is the sigmoid function.

$$f(X) = \begin{cases} 1, x > T \\ 0, x \leq T \end{cases}, \tag{2.8}$$

where T is the threshold level

$$f(X) = \frac{1}{1+e^{-x}}. \tag{2.9}$$

2.7.3 Multilayer Perceptron Networks

Multilayer perceptron (MLP) multilayer networks can solve more complicated problems than single-layer networks [8]. The term "multilayer" is used because this methodology is composed by several neurons arranged in different layers. Each connection between the input and hidden layers (or two hidden layers) is similar to a synapse (biological counterpart) and the input data is modified by a determined weight. Therefore, a three layer feed-forward network is composed by an input layer, two hidden layers and the output layer.

MLP is also called feed-forward neural networks because the data information flows only in the forward direction. In other words, the produced output of a layer is only used as input for the next layer. An important characteristic of feed-forward networks is supervised learning.

An important task in the MLP methodology is the training step. The training or learning step is a search process for a set of weight values with the objective of reducing/minimizing the squared errors of prediction. This phase is the slowest one and there is no guarantee of minimum global achievement. The most popular one is the back-propagation algorithm. This algorithm uses the error values of the output layer (prediction) to adjust the weight of layer connections. Therefore, this algorithm provides a guarantee of minimum (local or global) convergence.

The main challenge of MLP is the choice of the most suitable architecture. The speed and the performance of the MLP learning are strongly affected by the number of layers and the number of hidden unities in each layer. Figure 2.20 displays the influence of number of layers on the pattern recognition ability of neural network [27].

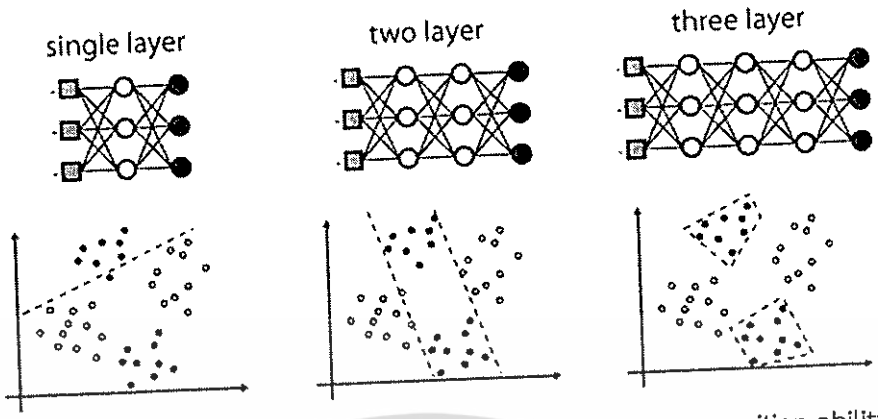

Figure 2.20 Influence of the number of layers on the pattern recognition ability. [27].

The increase in the number of layers in a MLP algorithm is proportional to the increase of complexity of the problem to be solved. The higher the number of hidden layers, the higher the complexity of the pattern recognition of the neural network.

2.7.4 Back-propagation

Artificial neural network are a supervised learning method [26]. Normally, a neural network requires a large training set of complete records to teach the model to improve the accuracy of the model just like a human learning process. As each observation from the training set is processed through the network, an output value is produced from the output node as in Figure 2.21.

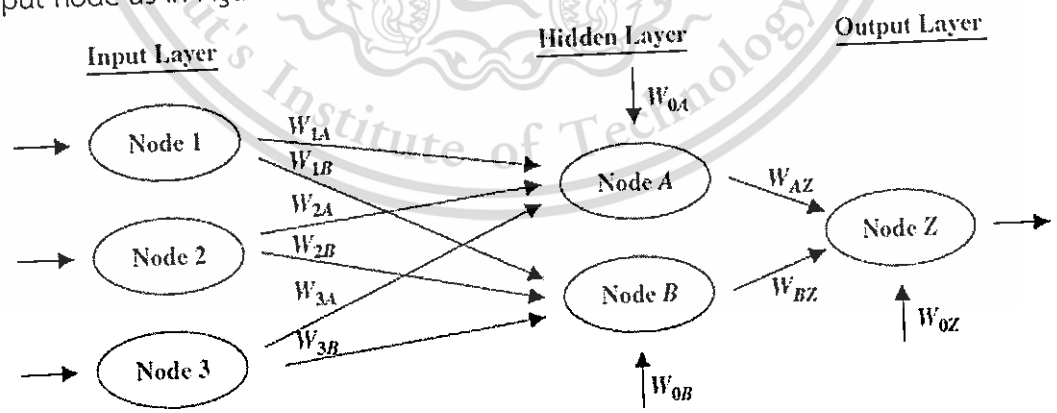

Figure 2.21 Simple neural network. [26]

This output value is then compared to the actual value of the target variable for this training set observation, and the error (actual−output) is calculated. Then the error

value of the output layer is used to adjust the new weight of layer connections. To measure how well the output predictions fit the actual target values, most neural network models use the sum of squared errors:

$$SSE = \sum_{records} \sum_{output\_nodes} (actual - output)^2 , \quad (2.10)$$

where the squared prediction errors are summed over all the output nodes and over all the records in the training set.

The problem is therefore to construct a set of model weights that will minimize the SSE. In this way, the weights are analogous to the parameters of a regression model. The "true" values for the weights that will minimize SSE are unknown, and our task is to estimate them, given the data. However, due to the nonlinear nature of the sigmoid functions permeating the network, there exists no closed-form solution for minimizing SSE as exists for least-squares regression.

2.7.5 Gradient Descent Method

Gradient-descent method [26] is used to find out the set of weights that will minimize SSE. Suppose that we have a set (vector) of m weights $w = w_0, w_1, w_2, \ldots, w_m$ in our neural network model and we wish to find the values for each of these weights that, together, minimize SSE. We can use the gradient descent method, which gives us the direction that we should adjust the weights in order to decrease SSE. The gradient of SSE with respect to the vector of weights w is the vector derivative:

$$\nabla SSE(w) = \left[\frac{\partial SSE}{\partial w_0}, \frac{\partial SSE}{\partial w_1}, \ldots, \frac{\partial SSE}{\partial w_m}\right], \quad (2.11)$$

that is, the vector of partial derivatives of SSE with respect to each of the weights.

To illustrate how gradient descent works, let us consider the case where there is only a single weight w_1. See Figure 2.22 that plots the error SSE against the range of values for w_1. We would prefer values of w_1 that would minimize the SSE. The optimal value for the weight w_1 is indicated as w_1^*. We would like to develop a rule that would help us move our current value of w_1 closer to the optimal value w_1^* as follows:

$$w_{new} = w_{current} + \Delta w_{current} \ , \tag{2.12}$$

where $w_{current}$ is the current location of w and $\Delta w_{current}$ is the change in the current location of w.

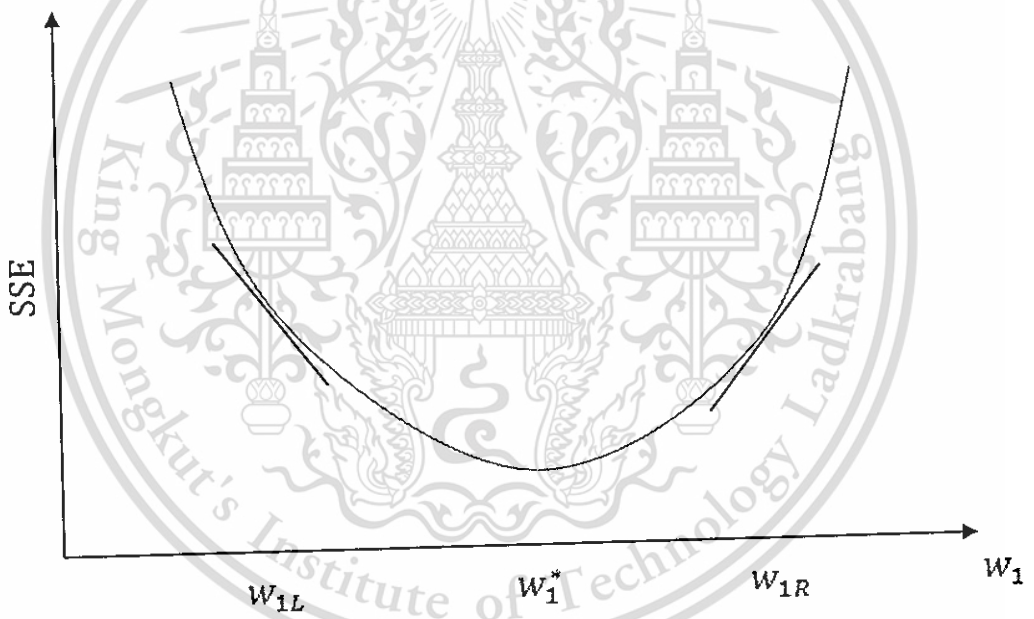

Figure 2.22 Using the slope of SSE with respect to w_1 to find weight adjustment direction.

Now, suppose that our current weight value $w_{current}$ is near w_{1L}. Then we would like to increase our current weight value to bring it closer to the optimal value w_1^*. On the other hand, if our current weight value $w_{current}$ were near w_{1R}, we would instead prefer to decrease its value, to bring it closer to the optimal value w_1^*. Now the derivative $\partial SSE/\partial w_1$ is the slope of the SSE curve at w_1. For values of w_1 close to w_{1L}, this slope is negative, and for values of w_1 close to w_{1R}, this slope is positive.

Hence, the direction for adjusting $w_{current}$ is the negative of the sign of the derivative of SSE at $w_{current}$, as shown in equation as follows:

$$-sign\left(\frac{\partial SSE}{\partial w_{current}}\right). \tag{2.13}$$

Now, how far should $w_{current}$ be adjusted? Suppose that we use the magnitude of the derivative of SSE at $w_{current}$. When the curve is steep, the adjustment will be large, since the slope is greater in magnitude at those points. When the curve nearly flat, the adjustment will be smaller, because less slope. Finally, the derivative is multiplied by a positive constant η (eta), called the learning rate, with values ranging between zero and 1. The resulting form of $\Delta w_{current}$ is as follows:

$$\Delta w_{current} = -\eta\left(\frac{\partial SSE}{\partial w_{current}}\right). \tag{2.14}$$

Meaning that the change in the current weight value equals negative a small constant times the slope of the error function at $w_{current}$.

2.7.6 Back-propagation Rules

The back-propagation algorithm [26] takes the prediction error (actual − output) for a particular record and feeds back the error to the network, assigning partitioned responsibility for the error to the various connections. The weights on these connections are then adjusted to decrease the error, using gradient descent. Using the sigmoid activation function and gradient descent, Mitchell [18] derives the back-propagation rules as follows:

$$w_{ij,new} = w_{ij,current} + \Delta w_{ij}\,, \tag{2.15}$$

$$\Delta w_{ij} = \eta \delta_j x_{ij} \,, \tag{2.16}$$

where η represents the learning rate and x_{ij} indicates the input index i and node j, and δ_j represents the responsibility for a particular error belonging to node j. The error responsibility is computed using the partial derivative of the sigmoid function with respect to net j and takes the following forms, depending on whether the node is in the output layer or the hidden layer. If node j is in output layer, δ_j is as follows:

$$\delta_j = output_j(1 - output_j)(actual_j - output_j) \,, \tag{2.17}$$

If node j is in hidden layer, δ_j is as follows:

$$\delta_j = output_j\left(1 - output_j\right) \sum_{downstream} W_{jk}\delta_j, \tag{2.18}$$

where $\sum_{downstream} W_{jk}\delta_j$ refers to the weighted sum of the error responsibilities for the nodes downstream from the particular hidden layer node.

2.7.7 Advantages of Artificial Neural Network Classifier

Artificial neural network are widely spread and used in everyday services, products and applications [32]. One of the greatest advantages of artificial neural network is their capability to learn from their environment. Moreover, there are many advantages of artificial neural network as described follows.

a) Can be applied to many problems, as long as there is some data.
b) Can be applied to problems, for which analytical methods do not yet exist
c) Can be used to model non-linear dependencies.
d) If there is a pattern, then artificial neural network should quickly work it out, even if the data is 'noisy'.
e) Always gives some answer even when the input information is not complete.

2.7.8 Disadvantages of Artificial Neural Network Classifier

However, for all artificial neural network advantages and positive aspects, the artificial neural network also has its disadvantage as describe follows.

a) Like with any data-driven models, they cannot be used if there is no or very little data available.

b) There are many free parameters, such as the number of hidden nodes, the learning rate, minimal error, which may greatly influence the final result.

c) Not good for arithmetics and precise calculations.

d) Artificial neural network do not provide explanations. If there are many nodes, then there are too many weights that are difficult to interpret.

2.8 C5.0 Classifier

C5.0 algorithm is an extension of C4.5 algorithm which is also extension of ID3. It is the classification algorithm which applies in big data set. It is better than C4.5 on the speed, memory and the efficiency [33]. Table 2.1 shows comparison between different decision tree algorithm [34].

Table 2.1 Comparisons between different decision tree algorithm

	ID3	C4.5	C5.0
Type of data	Categorical	Continuous and categorical	Continuous and categorical, date, times, timestamps
Speed	Low	Faster than ID3	Highest
Pruning	No	Pre-pruning	Pre-pruning
Boosting	Not supported	Not supported	Supported
Missing values	Can't deal with	Can't deal with	Can deal with
Formula	Use information entropy and information gain	Use split info and gain ration	Same as C4.5

The process of a decision tree is to find the best attribute variables of classification ability. Then through the best attribute variables, the data will be divided into multiple subsets. To each subset, the best property of the classification capability is found, which can be used to be divided into multiple sub-sets further. This is always an iterative operation until all subsets contain only one category or the number of samples is smaller than a certain threshold [5]. Finally, examine the lowest level split, those sample subsets that do not have remarkable contribution to the model will be rejected. C5.0 is easily handled the multi value attribute and missing attribute from data set [35].

The C5.0 algorithm uses the concept of information gain or entropy reduction to select the optimal split. Suppose that we have a variable X whose k possible values have probabilities $p1, p2, \ldots, pk$. What is the smallest number of bits, on average per symbol, needed to transmit a stream of symbols representing the values of X observed? The answer is called the entropy of X and is defined as follows [26]:

$$H(X) = -\sum_{j} P_j \log_2(p_j) . \tag{2.19}$$

Where does this formula for entropy come from? For an event with probability p, the average amount of information in bits required to transmit the result is $-\log_2 p$. For example, the result of a fair coin toss, with probability 0.5, can be transmitted using $-\log_2 p = 1$ bit, which is a zero or 1, depending on the result of the toss. For variables with several outcomes, we simply use a weighted sum of the $-\log_2(p_j)$'s, with weights equal to the outcome probabilities, resulting in Equation 2.19.

C5.0 uses this concept of entropy as follows. Suppose that we have a candidate split S, which partitions the training data set T into several subsets, $T_1, T_2, \ldots, T_k$. The mean information requirement can then be calculated as the weighted sum of the entropies for the individual subsets, as follows [26]:

$$H_S(T) = \sum_{i=1}^{k} P_i H_S(T_i) , \tag{2.20}$$

where p_i represents the proportion of records in subset i . We may then define our information gain to be gain$(S) = H(T) - H_s(T)$, that is, the increase in information produced by partitioning the training data T according to this candidate split S. At each decision node, C5.0 chooses the optimal split to be the split that has the greatest information gain, gain(S).

2.8.1 Advantages of C5.0 Classifier

C5.0 is one of top 10 algorithms in data mining [36].The main advantage of C5.0 is interpretability. In this research [37], C5.0 provides the best performance compare to ID3, and C4.5. Among all these classifiers C5.0 gives more accurate and efficient result. There many advantages of C5.0 that will be described as follows [35].

a) High accuracy and low memory usage.
b) C5.0 requires little data preparation. Other techniques often require data normalization, dummy variables need to be created and blank values to be removed.
c) C5.0 able to handle both numerical and categorical data. Other techniques are usually specialized in analyzing datasets that have only one type of variable.
d) C5.0 usually do not require long training times to estimate.
e) Able to handle multi-output problems.
f) C5.0 also offers the powerful boosting method to increase accuracy of classification.
g) C5.0 models are robust in the presence of problems such as missing data and large numbers of input fields.
h) In addition, C5.0 models tend to be easier to understand than some other model types, since the rules derived from Clementine the model have a very straightforward interpretation. The model in form of trees also can be visualized. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.

2.8.2 Disadvantages of C5.0 Classifier

However, for all C.50 advantages and positive aspects, the C.50 also has its disadvantage as describe follows.

a) C5.0 can create over-complex trees that do not generalize the data well. This is called overfitting. Mechanisms such as pruning, setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.

b) There are concepts that are hard to learn because C5.0 does not express them easily, such as XOR, parity or multiplexer problems.

c) C5.0 creates biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the C5.0.

2.9. LITERATURE REVIEW

2.9.1 Customer Failure Modes Prediction for Hard Disk Drive Using Neural Network Rank-Level Fusion

In this paper [38], key parameters from manufacturing are selected to predict head disk interaction (HDI) failure as show in Figure 2.23. In this paper, HDI related failure modes are studied. 512 samples are collected to create the model. PCA (Principle Component Analysis) is applied to screen for important key parameters. Applying PCA improved accuracy 2-3%. The author selected four classifiers: Neural Network (NN), Discriminant Analysis (DA), Bayesian Networks (BN), and Support Vector Machine (SVM). The total samples were split randomly to create a 60% training set and a 40% testing set. The author looked at the performance of all combinations of four classifiers through 3 different rank-levels: Borda Count, Logistic Regression, and Neural Networks Fusion. With Neural Networks Fusion method, the combination of NN + BN + SVM + DA shows the highest accuracy at 86.6%.

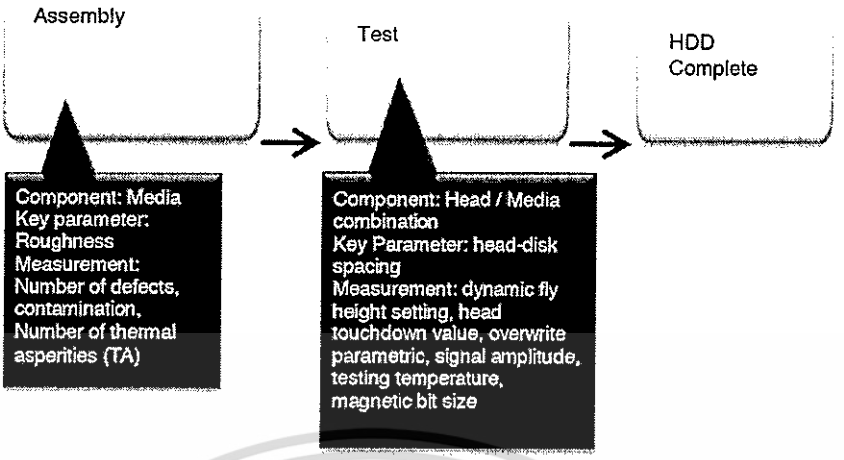

Figure 2.23 Key parameters and respective measurements taken during the manufacturing of HDD. [38]

2.9.2 Hard Disk Drive Failure Mode Prediction from SMART Attribute Using Data Mining Method

In this paper [39], the C5.0 algorithm can be used for the multicast classifier on mixed types of data with 93.03% accuracy for failure mode predication. Table 2.2 shows model results of each classification algorithm.

In this thesis, the software not only can help the analyst to group or sample drives for failure analysis (FA), but it also reduces turnaround time (TAT). The analyst will be able to find the root cause and the corrective action of the failure much faster by using the software. Moreover, it will also directly increase the quality and the reliability of the product.

The author will develop this software to analyze data for production, to screen some failed drives before sending drives to the customer. The author also will develop software to monitor hard disk health and performance at the customer site to alert when the drive has a failure warning, which will increase quality and reliability for production and the customer.

Table 2.2 Model results of each classification algorithm

Model	Accuracy
C5.0	93.03%
Neural Network	56.25%
Bayes Network	53.00%
SVM	42.03%

2.9.3 Study of Neural Network for Fly Height Failure Pattern Classification in Hard Disk Drive

For this thesis [8], three neural network (NN) models were studied and used for Fly Height (FH) failure pattern classification in hard disk drive (HDD) manufacturing process: learning vector quantization (LVQ), back propagation (BP), and probabilistic neural network (PNN).

In this thesis, the fly height failure pattern was studied. There are 6 types of failure patterns that are found in test process, as shown in Figure 2.24. In a FH profile, the y-axis is the heater power and the x-axis is the data zone of a disk going from the outer diameter toward the inner diameter of the disk.

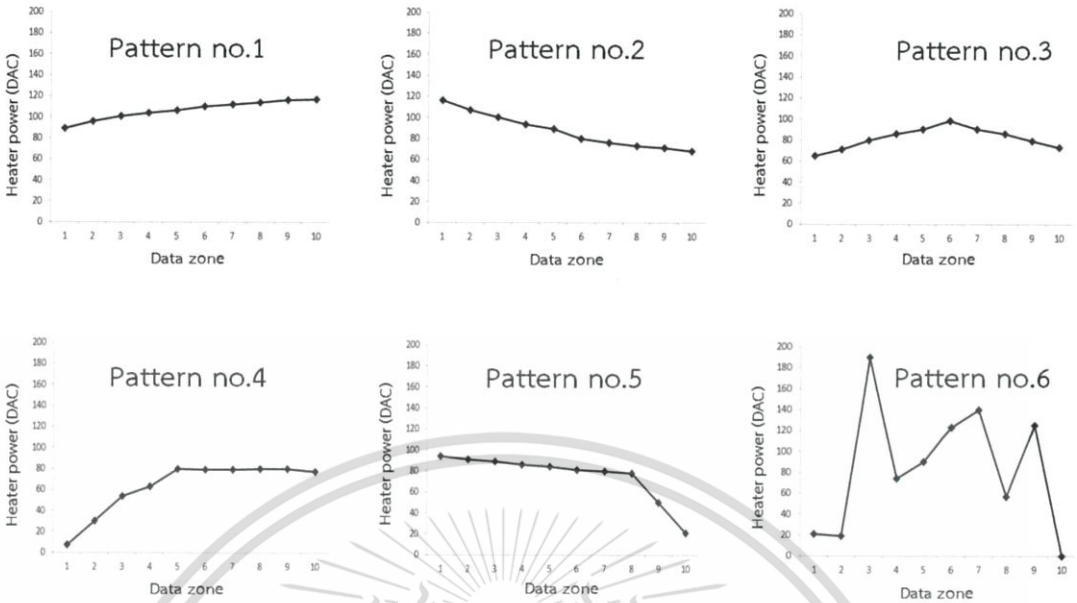

Figure 2.24 Six types of failure patterns from FH measurement. [8]

Based on this failure analysis data, each FH pattern can indicate one of three causes of failure. Pattern no.1 and no.2 indicate failures are from physical damage on head gimbal assembly (HGA). Patterns no.3, no.4 and no.5 relate to contamination on the slider air bearing surface (ABS). Pattern no.6 is head instability. Examples of failure analysis results are shown in Figure 2.25, 2.26, and 2.27.

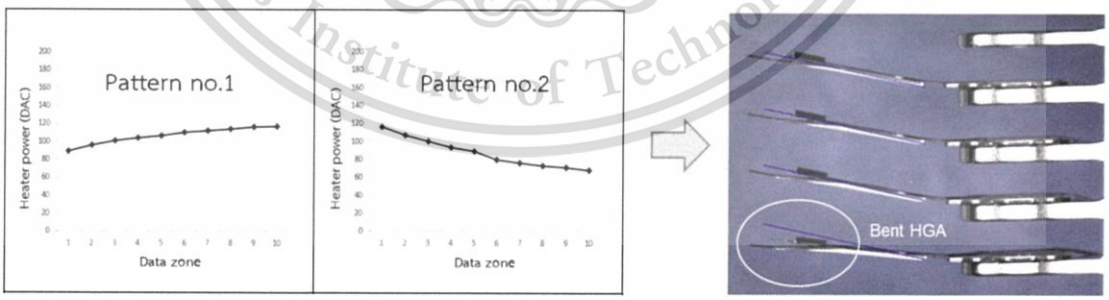

Figure 2.25 FA result of pattern no.1 and no.2. [8]

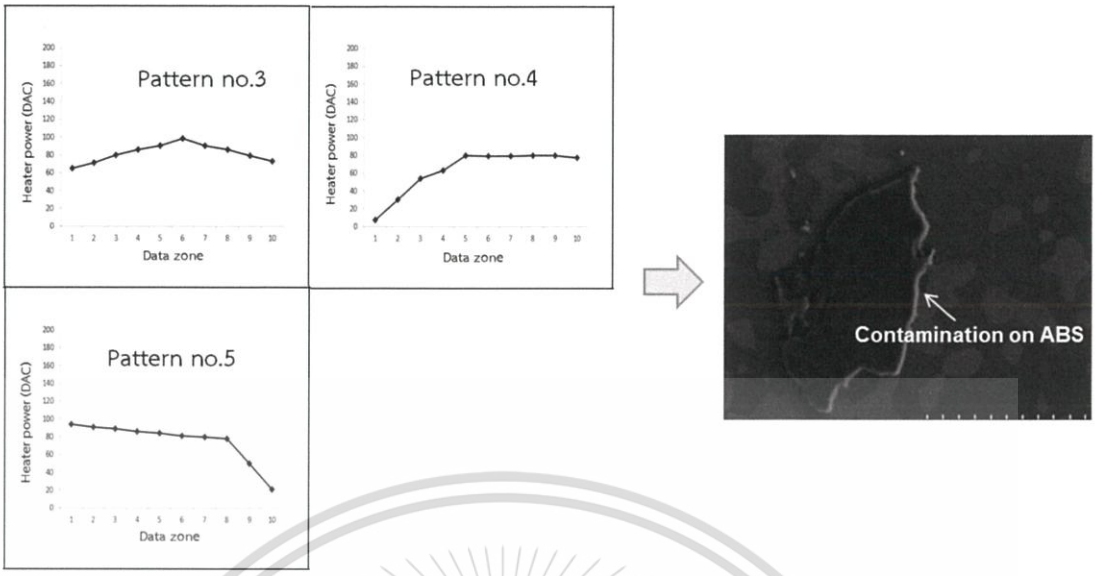

Figure 2.26 FA result of pattern no.3, 4, and 5. [8]

Figure 2.27 FA result of pattern no.6. [8]

The FA results of pattern no.1 and no.2, in Figure 2.25, show the side view of head stack assembly (HSA). The bent suspension is obviously seen as a root cause of failure. The second FA result of pattern no.3, 4, and 5, in Figure 2.26, is illustrated by an FESEM (Field Emission Scanning Electron Microscopy) image. There is a contamination on the face of the slider ABS. The last FA result of pattern no.6 in Figure 2.27 shows the read back signal from media with various write patterns. This is monitored by

oscilloscope, which is observed via a baseline popping noise. These abnormal cases are screened by FH testing before shipment to customers.

In this thesis, the author found that the PNN can achieve the highest average accuracy at 96.2% while LVQ reaches 90.7%. BP has the worst performance with the highest average accuracy of only 81.8%.

CHAPTER 3

RESEARCH METHODOLOGY

This research methodology consists of five steps including the measurement tool, data and sample collection, data pre-processing, and classification process.

The new software written in the python language was used during data-preparation. RapidMiner 5 and Clementine 12.0 applications were used in the analysis process. This chapter presents the steps of the measurement system and setup procedure, assumptions on materials and methods, how to collect and prepare the desired data, how to evaluate based on various conditions and the expected results of this experiment. A procedure for accurate measurement will also be presented.

3.1 The Measurement System

The HDD tester is the measurement tool used in this experiment. It can measures drive performance based on the test sequence identified in the test script. Each test sequence HDD is tested against certain specifications. If it meets all specifications, it can be shipped to the customer. The HDD tester keeps all information collected during the test process and accumulates it into one text document. The text document is called the "testing log".

3.1.1 Repeatability in Measurement System

With regards to the "Repeatability" study, we found that drives can pass the test process without any component replacement. A temporary fail may occur if the system generates noise and disturbance during the test. This fact told us that we can recover false failures by re-test or re-calibration.

3.1.2 Novel System for Recovering Failed Drive

Because of the repeatability study, the HDD factory sees an opportunity to recover false fail drives. The recovery process may help to change some properties for some drives and cause the test result to be different. This is a clear benefit to create a

new method that helps reduce the number of failing drives by reclaiming over-reject drives. This new method is an in-process recovery that is more effective than current methods because we are able to put additional certification/calibration during the test process that the traditional system does not have. The traditional approach only performs a simple re-test or recommends changing out parts/components (which can be costly). We call this method "intelligence Recovery" or iRecovery for short. The flow of "iRecovery" is shown in Figure 3.1.

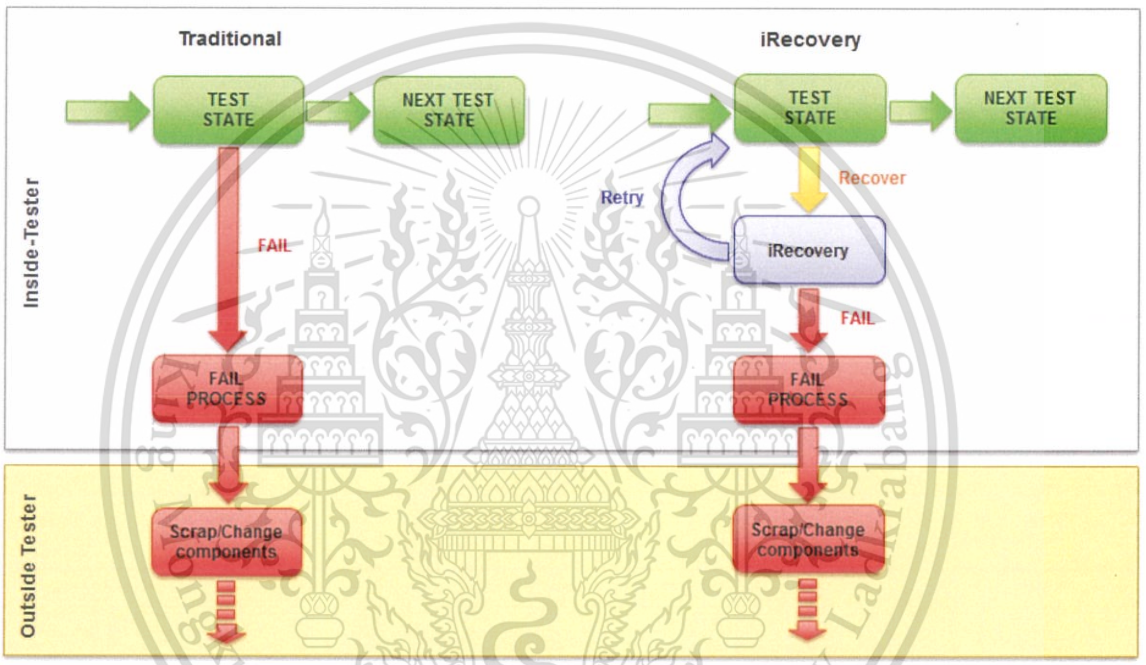

Figure 3.1 Work flow comparison between old vs. new method.

3.2 Data and Sample Collection

In this research, Data is collected from the actual test process in the form of a testing log. The testing log contains information about the drive performance which can be generalized into three basic categories: testing command, testing result, and debug message. The testing command is a low level command sent to the drive to test something (e.g. seek, read, and write). The testing result is the result from the test algorithm performed by the testing command. The debug message is a human readable message for analysis. Figure 3.2 shows three basic categories message in testing log.

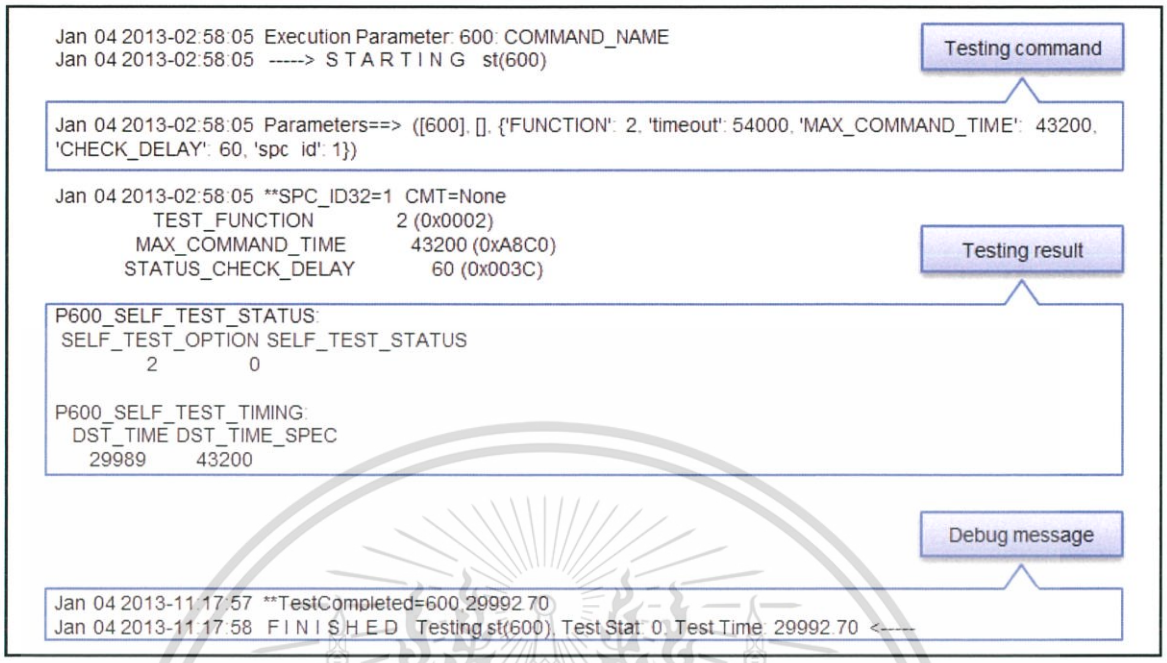

Figure 3.2 Three basic categories information in testing log.

To build the false pass classification model, testing logs were collected from "Pass" samples at test process. There are 4,142 samples of true pass drives and 21 samples of false pass drives. It is hard to find false pass drive samples because it is a kind of event that rarely occurs.

To build the false fail classification model, testing logs were collected from "Fail" samples at test process. There are 305 samples of true fail drives and 374 samples of false fail drives.

3.3 Data Analysis

Based on failure analysis of false fail drives, false fail drives can be mixed in with many failure symptoms. Therefore, we selected to study only the highest value failures; that is the failures with the highest recovery rate. This allows the factory to obtain more good/passing drives out of a failure symptom. The symptom called "delta fly height between the outer zone and inner zone is over a failure limit" is selected to study in this research.

Based on failure analysis of false pass drives, we choose to analyze only one failure symptom. The symptom is called "Passer from fail".

The analysis details of false fail drives and false pass drives will be described in the section named "Appearance of false fail drives" and "Appearance of false pass drives" as follows.

3.3.1 Appearance of False Fail Drives

In this research, we select to study drives where the delta fly height between the outer zone and inner zone is over a failure limit, because these failures have a high recovery rate. In the fly height measurement process, the spacing distance between recording head and media are considered in terms of the voltage values that are applied into the heater element of the recording head. It causes the writer and reader elements to protrude until they are in contact with the media, which is detected by the servo detector system. The heater voltage values are in the Digital to Analog Converter (DAC) unit for both read and write operations in each location (data zone) on the media. Data zones are separated into 3 groups: outer disc (OD), middle disc (MD), and inner disc (ID).

A typical false fail drive has a fly height profile such as pattern no. 4 (see Figure 2.26). The main cause of poor repeatability of failure drives comes from noise and disturbance during the test. Noise and disturbance can cause measurement errors. Figure 3.3 and Figure 3.4, shows the fly height profile of the failed head of a false fail drive. During the first run (red line) the drive failed because the delta fly height between the outer zone and inner zone the test's failure limit. Re-running the drive allowed it to pass with an acceptable/good fly height profile (green line).

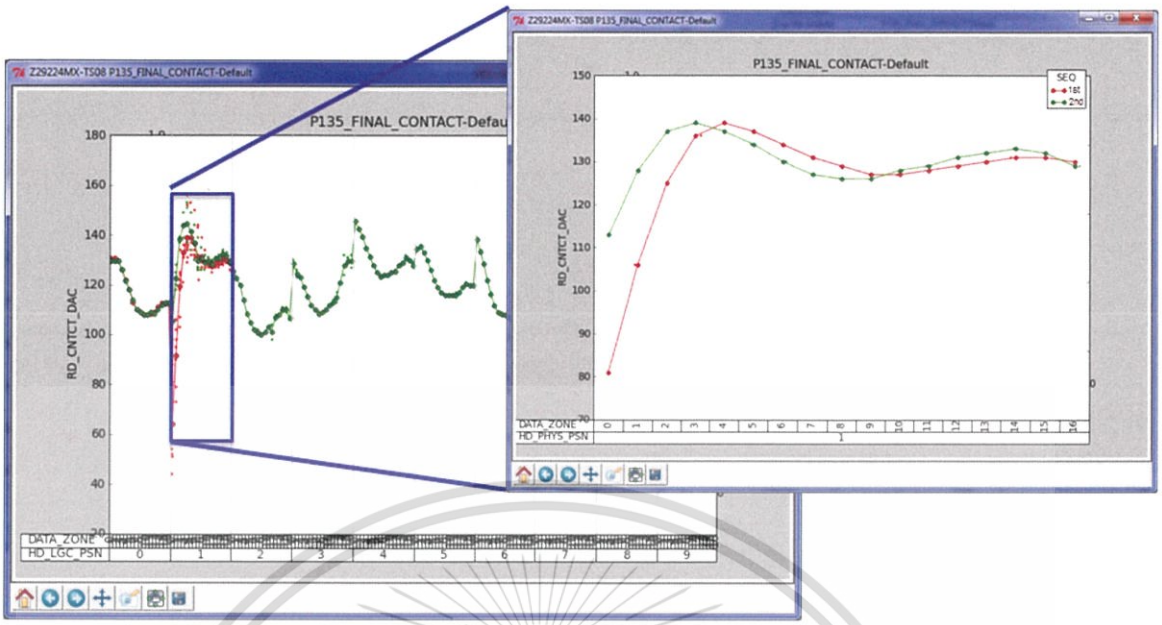

Figure 3.3 Example of false fail drive profile.

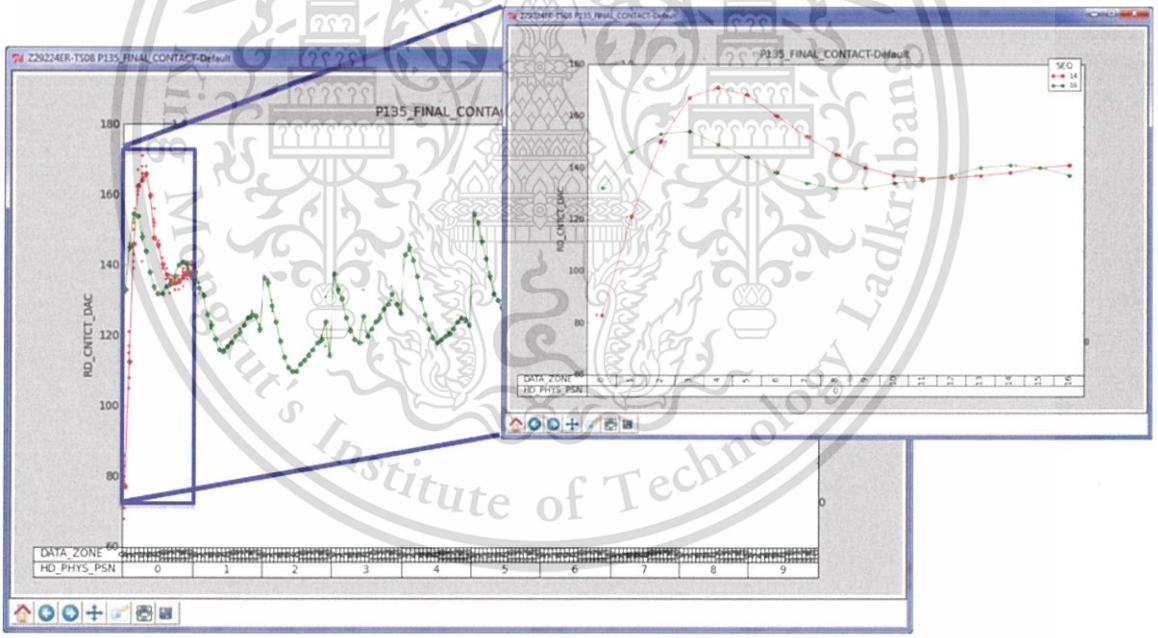

Figure 3.4 Example of false fail drive profile.

3.3.2 Appearance of False Pass Drives

A false pass drive can come from an excursion in the test process where the fail process suddenly stops testing the drive but does not report a failure message. A fail

process is a test state that is performed when a drive fails. A false pass drive skips some of the test process but still returns a passing status as shown in Figure 3.5. However, a false pass drive can be detected by a verification process that occurs after test process.

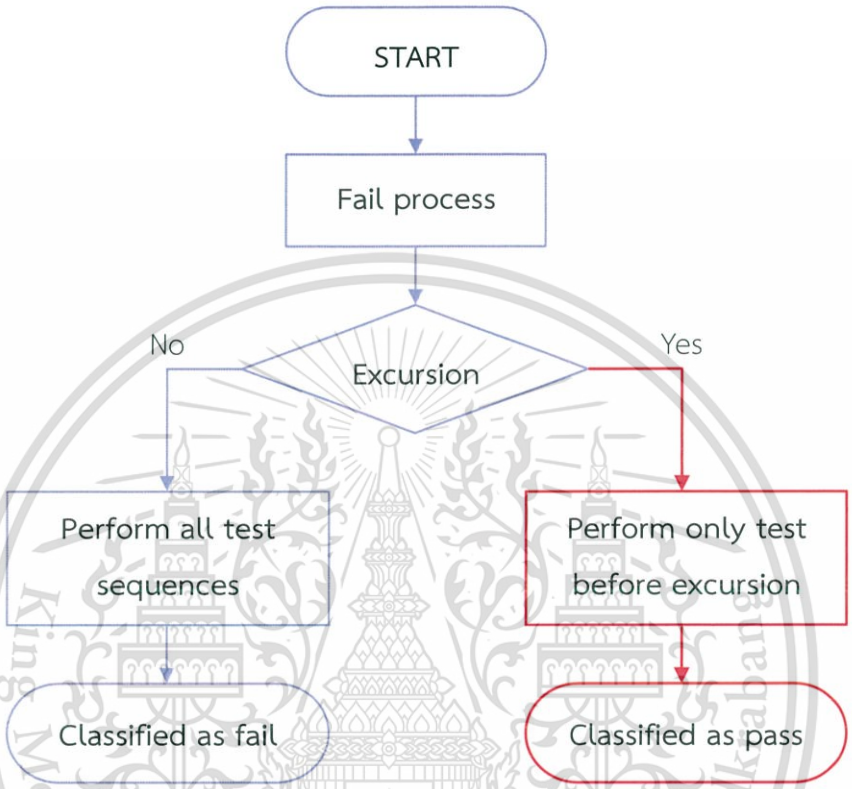

Figure 3.5 Flow chart of false pass drives.

Based on failure analysis, a false pass drive was shown to have skipped some tests in the fail process. However, it is difficult to detect false pass automatically by machine because a lot of information (some irrelevant) is mixed in the testing log

3.4 Data Pre-processing

Because of differences between false pass and false fail failure characteristic, we have to use two different classification methods. Data pre-processing depends on the classification method.

3.4.1 Data Pre-processing for False Pass Drives

In order to establish a methodology to classify the false pass portion, Text classification was applied. Input data is a file that contains text information. In the literature [1], it is shown that filtering data can improve the accuracy and effectiveness of classifying algorithms. The data filtering process is used to eliminate redundant information and reduce unnecessary data. To reduce unnecessary data, the new software written in the python language was used in the data transformation process. Data transformation is applied to convert a set of data values from the data format of a source data system into the data format of a destination data system. The use of regular expressions with arguments [40], a programming technique for locating specific character strings embedded in character text, would allow all instances of a particular pattern to be replaced with another pattern using parts of the original pattern.

For example, original data shown in Figure 3.6 could both be transformed into a more compact form as shown in Figure 3.7.

```
Jan 04 2013-02:58:05  Execution Parameter: 600: COMMAND_NAME
Jan 04 2013-02:58:05  -----> S T A R T I N G  st(600)
Jan 04 2013-02:58:05  Parameters==>  ([600], [], {'FUNCTION': 2, 'timeout': 54000, 'MAX_COMMAND_TIME': 43200, 'CHECK_DELAY':
60, 'spc_id': 1})
Jan 04 2013-02:58:05  **SPC_ID32=1  CMT=None
            TEST_FUNCTION          2 (0x0002)
          MAX_COMMAND_TIME         43200 (0xA8C0)
        STATUS_CHECK_DELAY          60 (0x003C)
P600_SELF_TEST_STATUS:
 SELF_TEST_OPTION SELF_TEST_STATUS
           2              0
P600_SELF_TEST_TIMING:
  DST_TIME DST_TIME_SPEC
    29989        43200
Jan 04 2013-11:17:57  **TestCompleted=600,29992.70
Jan 04 2013-11:17:58  F I N I S H E D  Testing st(600), Test Stat: 0, Test Time: 29992.70  <----
```

Figure 3.6 The data format of a source data system.

```
StParam([600], [], {'FUNCTION': 2, 'timeout': 54000, 'MAX_COMMAND_TIME': 43200, 'CHECK_DELAY': 60, 'spc_id': 1})
```

Figure 3.7 The data format of destination data system.

In other words, longer format of command calling would be replaced with a shorter format using all of the original set of arguments. After passing the data filtering process, data will be in text format and ready for text classification classifiers.

3.4.2 Data Pre-processing for False Fail Drives

In order to establish a methodology to classify the false fail drives, statistical classification was applied. Statistical classification input is a set of properties but our data is in the form of a testing log. Because the testing log contains all of the drive test information resulting in a large text document, data pre-processing is used to retrieve only the interesting/applicable information and gather it into one document. New pre-processing software specifically written for this study was created. This filtered each of the results files, retrieved the target information and transformed that information into records in an Excel file. The final Excel file contains multiple records that are information from individual drives. For example, information of 100 drives will be transformed into 100 records. Each record contains the interesting properties of each drive. Each property is represented in one column as show in Figure 3.8. The data in Figure 3.8 is only an example. It is not actual data.

SN	STATUS	Property 1	Property 2	Property 3	Property 4	Property 5
Drive No.1	True fail	2	10	10	R	1
Drive No.2	True fail	0	10	11	W	1
Drive No.3	True fail	0	10	12	W	1
Drive No.4	True fail	3	10	13	W	1
Drive No.5	True fail	6	10	14	R	1
Drive No.6	False fail	9	10	15	W	1
Drive No.7	False fail	1	10	16	R	1
Drive No.8	False fail	8	10	17	R	1
Drive No.9	False fail	0	10	18	W	1

Figure 3.8 False fail classification input.

We then sent selected interesting properties as input of the classification algorithm. See Table 3.1.

Table 3.1 Interesting properties from testing log

Description	Type
Failed Head number.	{1,2, ... 10}
Temperature upon finding fly height.	Continuous
Mean temperature of the whole test process.	Continuous
Name of sensor that failed. Reader or Writer	{"R","W"}
Count of sensor that failed in the same mode.	Continuous
Count of retries in finding fly height process.	{1,2,3,4}
Error code of each retry.	Continuous
Contact DAC of each zone.	Continuous

3.5 Classification Process

To explain classification process, two sub-topics: classification system, and classification evaluation will be presented.

3.5.1 Classification System

The classification system used in this research is separated into 2 systems. First is a classification system for false pass drives. Second is a classification system for false fail drives.

In the classification system for classifying false pass drives, empirical data were filtered to primary testing commands such as read/write command, which were then applied to machine-learning based algorithms. There are a lot of parameters in passing drives, and selecting parameters to a small sub-set of parameters would be a limitation. As such, we need to add all test parameters to the prediction model. There are clear benefits to using text classification which can keep all parameters and apply machine-learning based algorithms. False pass drive classification system as shown in Figure 3.9.

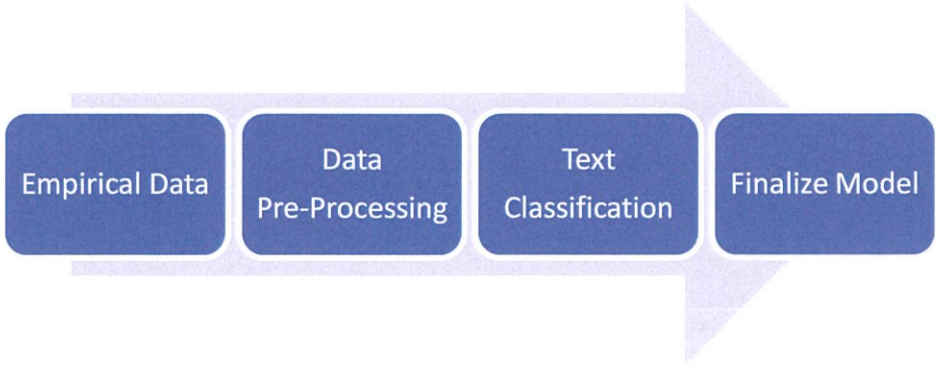

Figure 3.9 False pass drive classification system.

In the classification system for classifying false fail drives, empirical data were filtered to a set of related parameters, and then machine-learning based algorithms were applied. False fail drive classification system as shown in Figure 3.10

Figure 3.10 False fail drive classification system.

3.5.2 Classification Evaluation

Accuracy is also used as a statistical measure of how well a binary classification test correctly identifies. The accuracy is the proportion of true results (both true positives and true negatives) among the total number of cases examined [41, 42].

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}, \quad (3.1)$$

where TP is true positive (classify as 1 when the sample test result is 1), TN is true negative (classify as 0 when the sample test result is 0), FP is false positive (classify as 1 when the sample test result is 0), and FN is false negative (classify as 0 when the sample test result is 1). An accuracy of 100% means that the measured values are exactly the same as the given values.

In this thesis, the test options used for evaluating machine learning algorithms are multiple split tests [43]. The concept of multiple split tests is split test getting different results on different splits of the dataset is to reduce the variance of the random process and do it many times. We can collect the results from a fair number of runs (10 for this thesis) and take the average.

For example, we split our dataset 90%/10%, ran our algorithm and got an accuracy and we did this 10 times with 10 different splits. We might have 10 accuracy scores as follows: 87, 87, 88, 89, 88, 86, 88, 87, 88, and 87. The average performance of our model is 87.5.

CHAPTER 4

EXPERIMENT AND RESULTS

This chapter presents the evaluation and performance comparison of false pass and false fail classification, which is associated to classifiers and settings. The topics include: (1) artificial neural network classifier experiments, (2) C5.0 classifier experiments, (3) *k*-NN classifier experiments, and (4) the naïve Bayes classifier experiments. Topic (1) and (2) were used to classify false fail drives. Topic (3) and (4) were used to classify false pass drives.

4.1 Artificial Neural Network Classifier Experiments

For all following experiments, the model is built based on the "artificial neural network" classifier developed in Clementine 12.0 [44]. Figure 4.1 illustrates the artificial neural network overall setting in Clementine 12.0 used to create prediction model and then measure prediction performance.

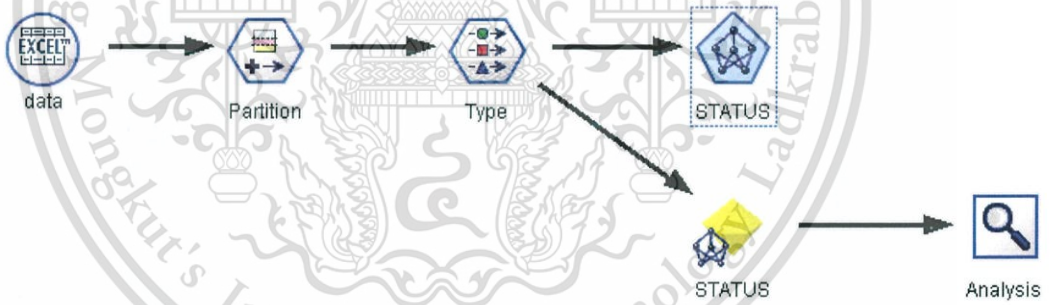

Figure 4.1 The neural network overall setting in Clementine 12.0.

The goal of experiment is to find highest accuracy model. 305 true fail and 374 false fail drives testing data were collected to be used to build the output model. The artificial neural network experiments include: (1) training set size experiment, (2) hidden layer one size experiment, (3) hidden layer two size experiment, (4) alpha value experiment, and (5) the learning rate experiment.

4.1.1 Training Set Size Experiment

Suppose we have L number of training samples variant from 10% to 90% of all data samples. The networks are trained with these samples and the number of testing sample is equal to (100 – L) %.

In this experiment, training set size and training cycles are varied and average accuracy is observed. The default setting of the Clementine 12.0 application training cycle was set to 250 cycles. The result is shown in Figure 4.2.

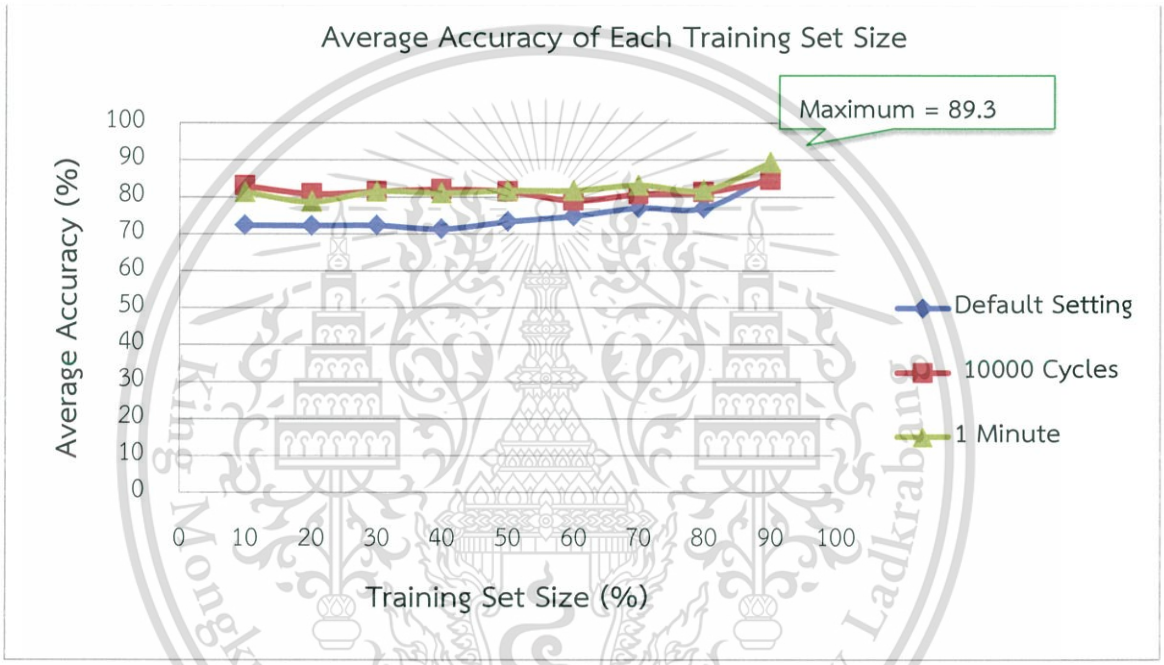

Figure 4.2 Average accuracy of each training set size.

From experiment result in Figure 4.2, the line graph represents the average accuracy increases with increasing training set size. The average accuracy also increases with increasing training cycle loop.

In clementine 12.0 application has the option to fix training time instead of fixing training cycles. We set the experiment to train the model for 1 minute and monitored its behavior. The average accuracy was increased following the training time until it became stable around the 1 minute mark. Even if we continue training more than 1 minute, average accuracy will not increase any further as shown in Figure 4.3, the X-

axis represents the average accuracy, while the Y-axis represents training time. In Figure 4.3, we capture training time for 60 seconds.

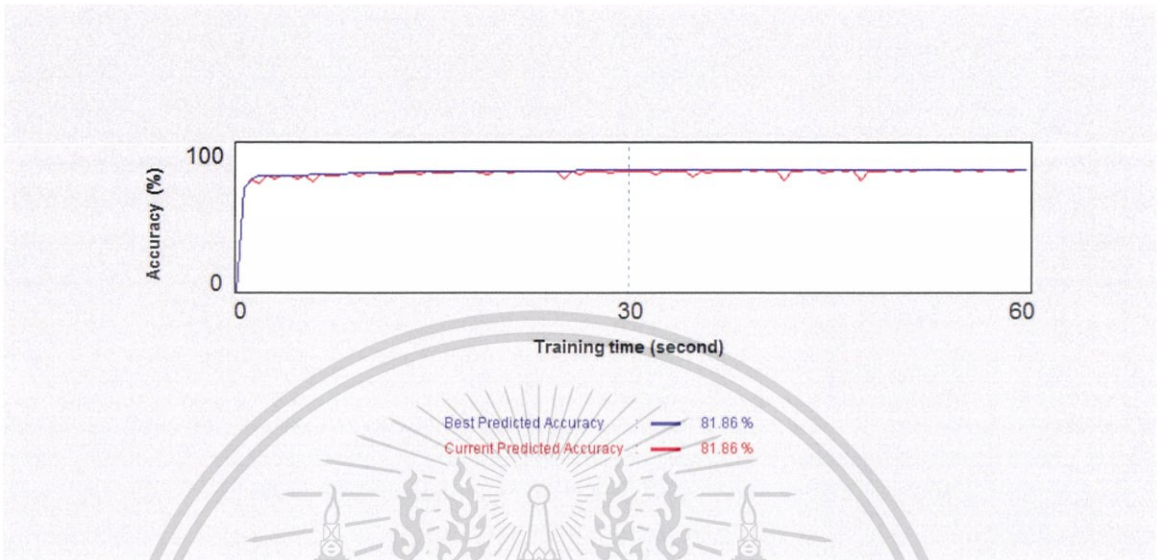

Figure 4.3 Accuracy profiles in 1 minute training time.

From accuracy profiles in 1 minute training time graph, the graph represents the average accuracy increased quickly in few seconds then slightly increased until stable at around 30 seconds.

The training set size experiment on Figure 4.2 results show that 90% training set and 1 minute training time provides the best performance with an average accuracy at 89.3%. Therefore, 90% training set and 1 minute training time is selected for all following the artificial neural network classifier experiments.

4.1.2 Hidden Layer One Nodes Experiment

In this experiment, hidden layer one nodes is varied and average accuracy is observed. Average accuracy of each hidden layer one nodes is shown in Figure 4.4.

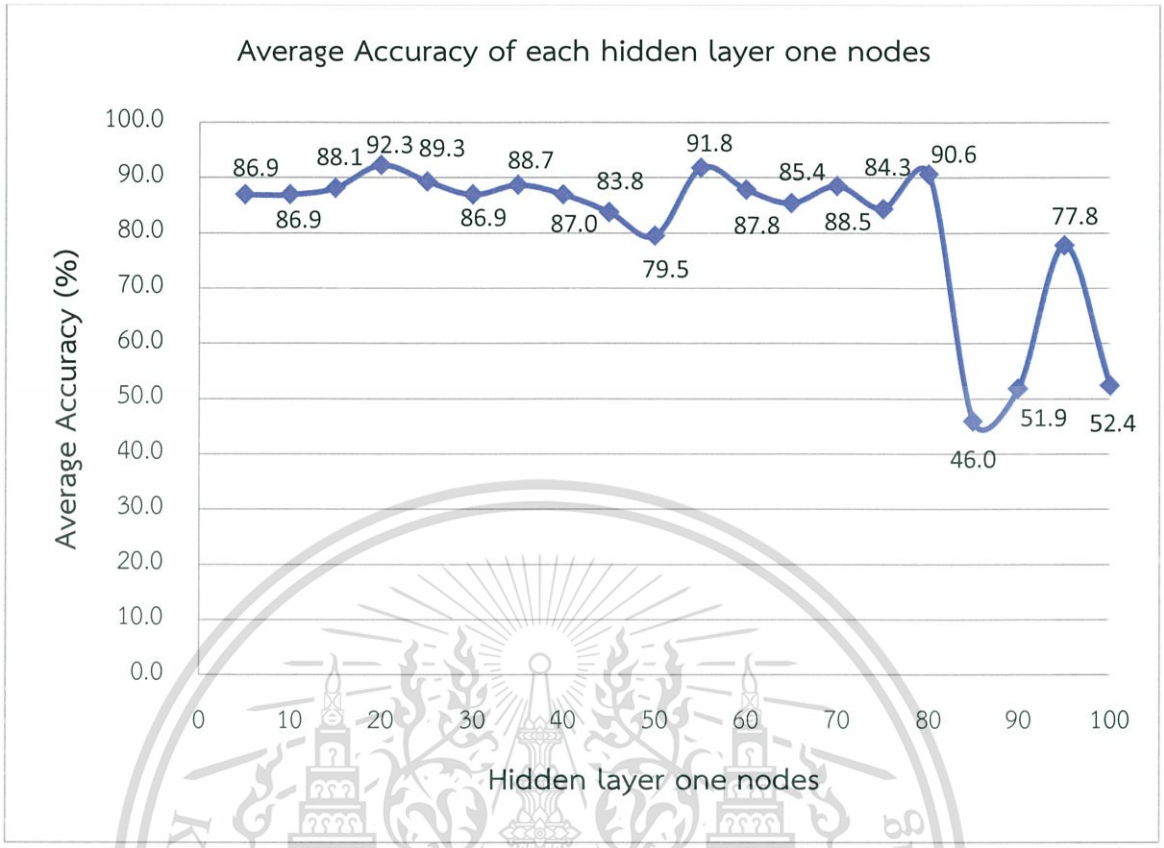

Figure 4.4 Average accuracy of each hidden layer one nodes.

The hidden layer one nodes experiment in Figure 4.4 result shows that the model which contains 20 nodes of hidden layer one provides the best performance with an average accuracy at 96.7%.

From hidden layer one nodes experiment overall perspective, the graph represents increasing of hidden layer one nodes from 5 to 80 is no significant improvement in the average accuracy, just random fluctuation. Increasing hidden layer one nodes from 85 to 100, however, shows decreasing in the average accuracy significantly.

The reason the average accuracy decreases when too many nodes are in the hidden layer one is called overfitting [45]. Overfitting occurs when the neural network has so much information processing capacity that the limited amount of information contained in the training set is not enough to train all of the nodes in the hidden layer. A second problem can occur even when the training data is sufficient. Too large a number of nodes in the hidden layer can increase the time it takes to train the network.

The amount of training time can increase to the point that it is impossible to sufficiently train the neural network.

4.1.3 Hidden Layer Two Nodes Experiment

In this experiment, the number of hidden layer two nodes is varied and average accuracy is observer. Average accuracy of each hidden layer two nodes experiment is shown in Figure 4.5.

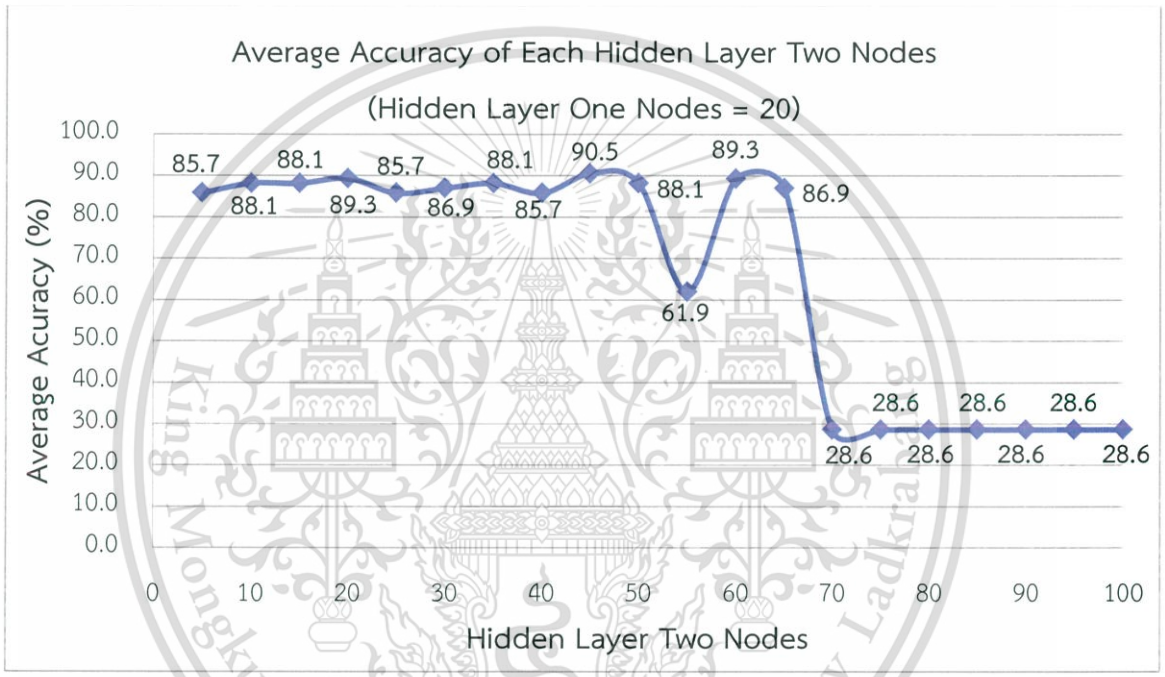

Figure 4.5 Average accuracy of each hidden layer two nodes.

The hidden layer two nodes experiment in Figure 4.5 result shows that hidden layer two nodes 45 provides the best performance with an average accuracy at 90.48%. However, the hidden layer two performance is still lower than using the hidden layer one only.

From hidden layer two nodes experiment overall perspective, the graph represents increasing the hidden layer two nodes from 5 to 65 is no significant improvement in the average accuracy. However, hidden layer two nodes from 70 to 100 shows a dramatically decrease in the average accuracy.

The reason the average accuracy decreases when too many nodes are in the hidden layer two is also overfitting (same as the overfitting that occurs when too many nodes are in hidden layer one).

4.1.4 Alpha Value Experiment

The meaning of alpha setting is a momentum term α used in updating the weights during training. Momentum tends to keep the weight changes moving in a consistent direction. The alpha setting is a value between 0 and 1. Higher values of alpha increase momentum, decreasing the tendency to change direction based on local variations in the data. Figure 4.6 shows the result of each alpha value setting.

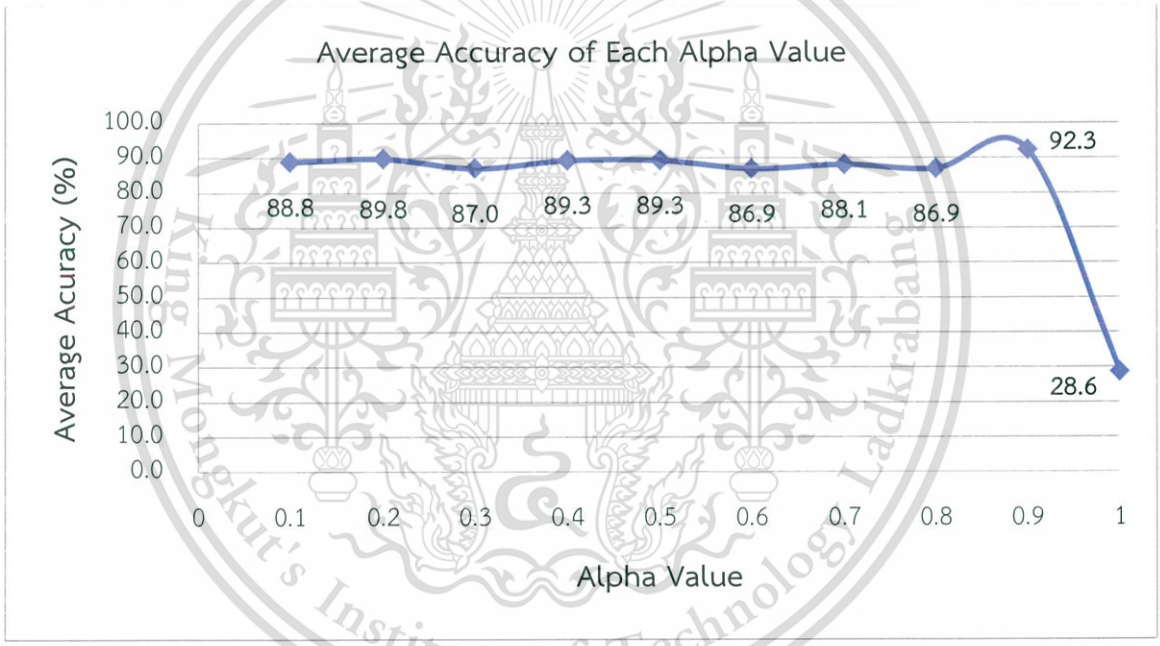

Figure 4.6 Average accuracy of each alpha value setting.

The alpha value experiment in Figure 4.6 result shows that alpha value 0.9 provides the best performance with an average accuracy at 92.3%.

From alpha value experiment overall perspective, the graph represents no significant improvement in the average accuracy just random fluctuation while alpha value less than 0.9, then dramatically decrease at alpha value equal to 1.

To make clearer about how the momentum term work [26], consider Figures 4.7 and 4.8. In both figures, the weight is initialized at location I, local minima exist at

locations A and C, with the optimal global minimum at B. In Figure 4.7, suppose that we have a small value for the momentum term α, symbolized by the small mass of the "ball" on the curve. If we roll this small ball down the curve, it may never make it over the first hill, and remain stuck in the first valley. That is, the small value for α enables the algorithm to easily find the first trough at location A, representing a local minimum, but does not allow it to find the global minimum at B.

Next, in Figure 4.8, suppose that we have a large value for the momentum term α, symbolized by the large mass of the "ball" on the curve. If we roll this large ball down the curve, it may well make it over the first hill but may then have so much momentum that it overshoots the global minimum at location B and settles for the local minimum at location C.

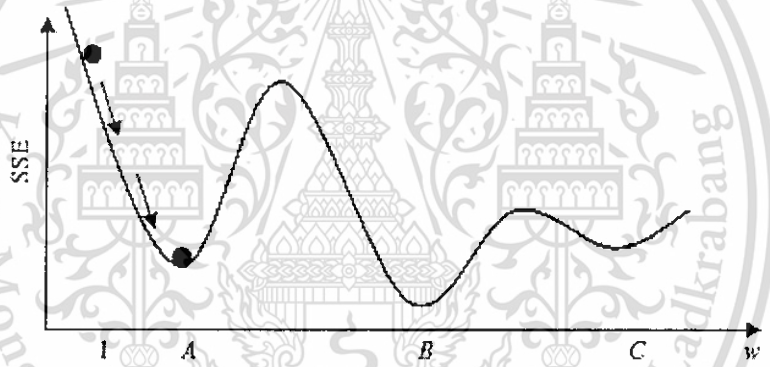

Figure 4.7 Small momentum may cause algorithm to undershoot global minimum. [26]

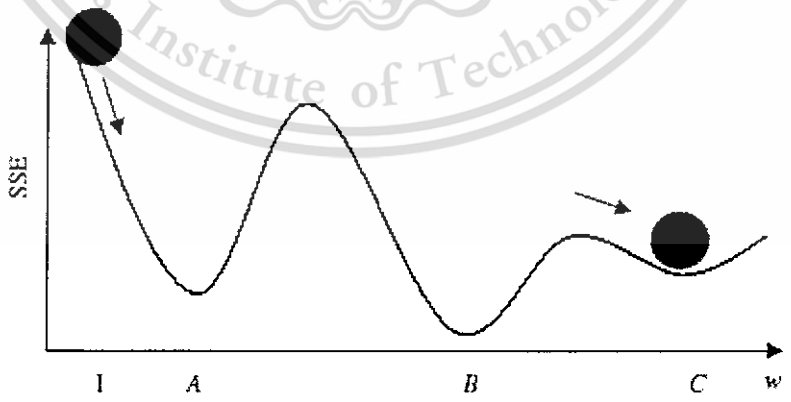

Figure 4.8 Large momentum may cause algorithm to overshoot global minimum. [26]

Thus, one needs to consider carefully what values to set for both the learning rate and the momentum term. Experimentation with various values of learning rate and the momentum term may be necessary before the best results are obtained. That is the reason why we did many experiments with various value of α until we saw that when the α = 0.9 was used, we can obtain best average accuracy.

4.1.5 The Learning Rate Experiment

Eta, the learning rate, controls how much the weights are adjusted at each update. Eta changes as training proceeds. Initial eta is the starting value of eta. During training, eta starts at initial eta, decreases to low eta, then is reset to high eta and decreases to low eta again. The last two steps are repeated until training is complete. This process is shown in Figure 4.9.

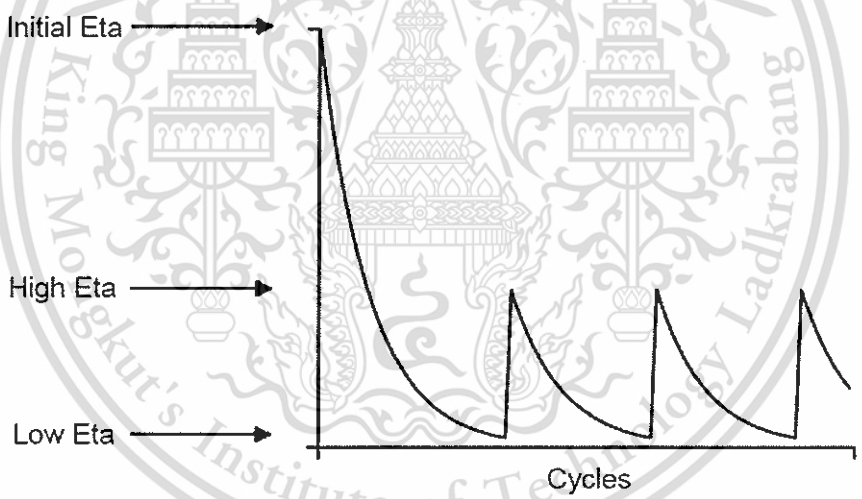

Figure 4.9 How eta changes during neural network training. [26]

In this experiment, high eta value is varied and the average accuracy is observed. Figure 4.10 shows average accuracy of each high eta value.

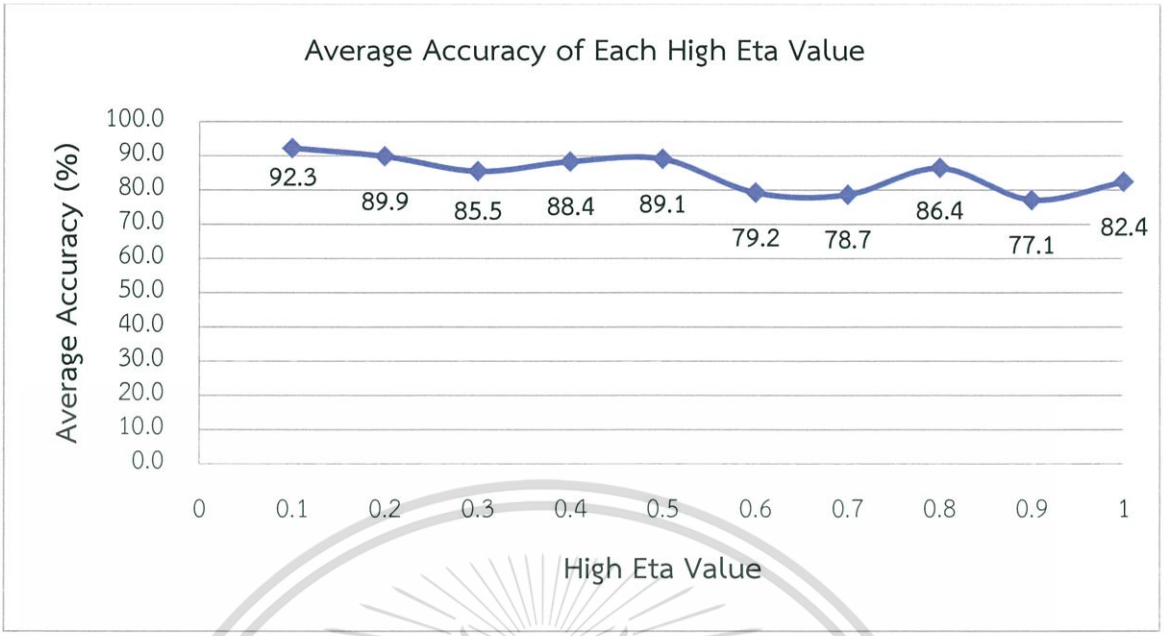

Figure 4.10 Average accuracy of each high eta setting value.

The high eta value experiment in Figure 4.10 result shows that high eta value 0.1 provides the best performance with an average accuracy at 92.3%.

From a high eta value experiment overall perspective, the average accuracy trend to decreases when the high eta value increases. The reason of decreasing in average accuracy when the initial eta value increases are large learning rate will tend to make the algorithm overshoot the optimal solution.

Another experiment about learning rate is the initial eta experiment. In this experiment, initial eta is varied and the average accuracy is observed. Figure 4.11 shows the result of each initial Eta value.

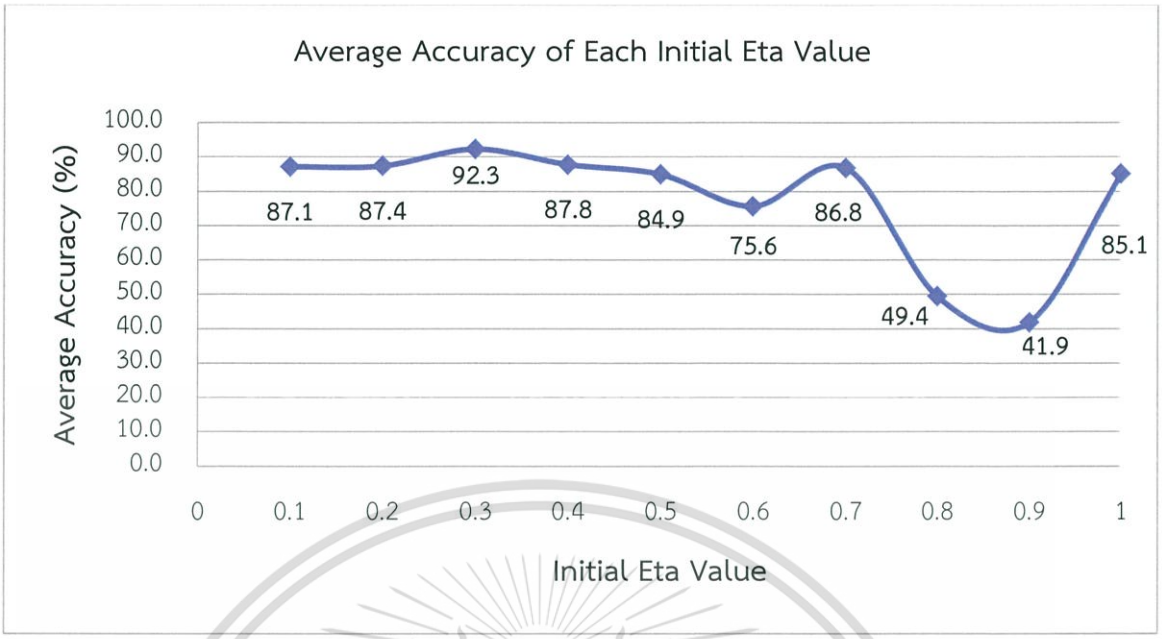

Figure 4.11 Average accuracy of each initial eta setting value.

The initial eta value experiment in Figure 4.11 result shows that an initial eta value 0.3 provides the best performance with an average accuracy at 92.3%.

From initial eta value experiment overall perspective, the average accuracy decreases when the initial eta value increases. The reason of decreasing in average accuracy when the initial eta value increases are large learning rate will tend to make the algorithm overshoot the optimal solution. However, the small learning rate is not always good. When the learning rate is very small, the weight adjustments tend to be very small. Thus, if learning rate is small when the algorithm is initialized, the network will probably take an unacceptably long time to converge. That is why the too small value of learning rate, such as 0.1 and 0.2, not provides the best result in this experiment.

4.2 C5.0 Classifier Experiments

The following experiments use a model built based on the "C5.0" classifier developed in Clementine 12.0. Figure 4.12 illustrates the C5.0 overall setting in Clementine 12.0 used for creating a prediction model and then measuring prediction performance.

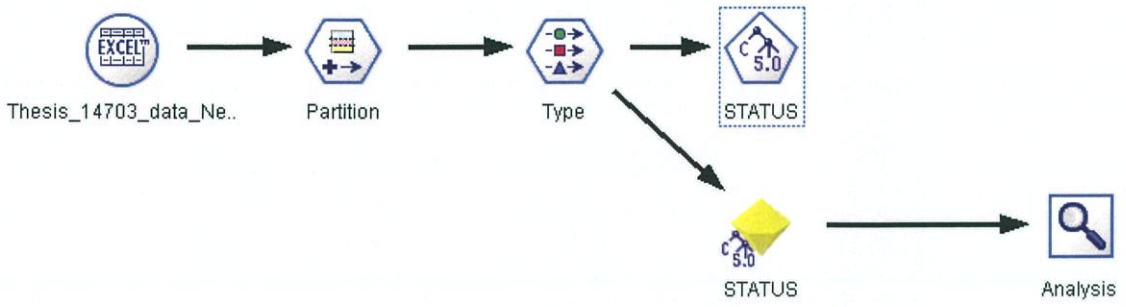

Figure 4.12 The C5.0 overall setting in Clementine 12.0.

The goal of this experiment is to find the highest accuracy model. 305 true fail and 374 false fail drives testing data were collected to be used for building the output model. The C5.0 classifier experiments include: (1) minimum records per child branch setting value experiment, (2) pruning severity setting value experiment, and (3) number of boosting iteration setting value experiment.

4.2.1 Minimum Records Per Child Branch Experiment

Minimum records per child branch is the size of subgroups that can be used to limit the number of splits in any branch of the tree. A branch of the tree will be split only if two or more of the resulting sub-branches would contain at least this many records from the training set. The default value is 2. Increase this value to help prevent overtraining with noisy data. In this experiment, the minimum records per child branch size is varied and average accuracy is observed. Average accuracy of each minimum records per child branch value is shown in Figure 4.13.

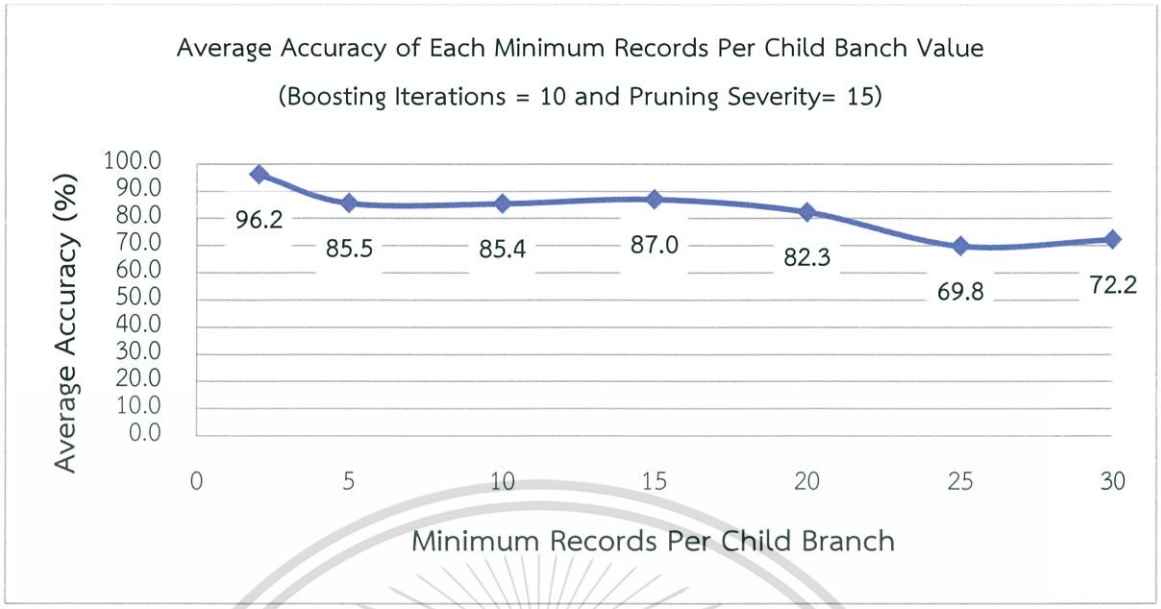

Figure 4.13 Average accuracy of each minimum records per child branch value.

The minimum records per child branch experiment in Figure 4.13 result shows that minimum records per child branch value 2 provides the best performance with an average accuracy at 96.2%.

From a minimum records per child branch experiment overall perspective, the average accuracy tends to decrease when minimum records per child branch increases.

Minimum records per child branch is the size of subgroups can be used to limit the number of splits in any branch of the tree. It prevents further splitting of a node that has reached the specified minimal size when the initial decision tree is being built. Low minimum records per child branch will increase accuracy but the model will become more complex, with a larger decision tree that has higher time and memory costs.

4.2.2 Number of Boosting Iteration Experiment

The C5.0 algorithm has a special method for improving its accuracy rate, called boosting. It works by building multiple models in a sequence. The first model is built in the usual way. Then, a second model is built in such a way that it focuses on the records that were misclassified by the first model. Then a third model is built to focus on the second model's errors, and so on. Finally, cases are classified by applying the whole set of models to them, using a weighted voting procedure to combine the

separate predictions into one overall prediction. Boosting can significantly improve the accuracy of a C5.0 model, but it also requires longer training. The Number of boosting iteration options allows you to control how many models are used for the boosted model. In this experiment, the number of boosting iterations is varied and average accuracy is observer. Average accuracy of each number of boosting iteration setting value is shown in Figure 4.14.

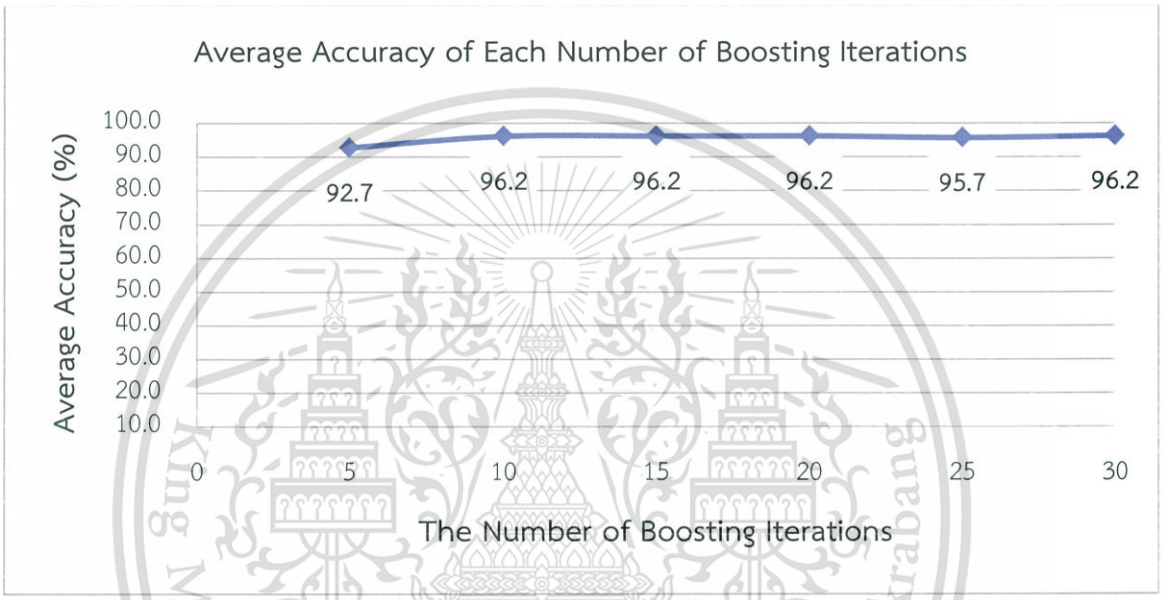

Figure 4.14 Average accuracy of each number of boosting iteration setting value.

The number of boosting iteration experiment in Figure 4.14 result shows that a number of boosting iteration value 10 provides the best performance with an average accuracy at 96.2% and smaller decision tree or rule set, more concise tree.

From the number of boosting iteration experiment overall perspective, the average accuracy tends to increase when the number of boosting iterations also increases and became stable at boosting iteration value equal to 10.

4.2.3 Pruning Severity Experiment

Pruning severity, determines the extent to which the decision tree or rule set will be pruned. Increase this value to obtain a smaller, more concise tree. Decrease it to obtain a more accurate tree. In this experiment, pruning severity is varied and average

accuracy is observer. Average accuracy of each pruning severity setting value is shown in Figure 4.15.

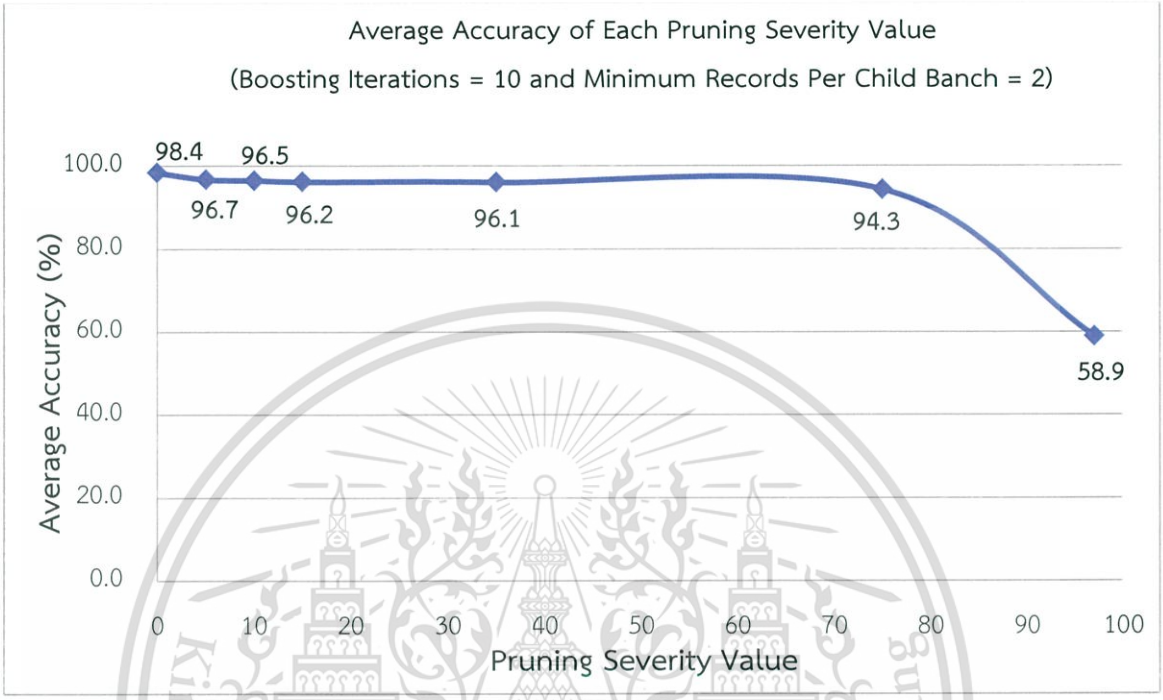

Figure 4.15 Average accuracy of each pruning severity value.

The pruning severity experiment in Figure 4.15 result shows that pruning severity value 0 provides the best performance with an average accuracy at 98.4%.

From pruning severity experiment overall perspective, the average accuracy trend to decreases when pruning severity increases. However, increase pruning severity value to obtain a smaller decision tree or rule set, more concise tree. It helps to minimize system resource consumption.

4.3 *k*-NN Classifier Experiments

To classify the false pass portion, a testing log was collected from a "Pass" sample at test process. In this research, 4,142 samples of true pass and 21 samples of false pass drives testing log were collected to be used for built the output model. The *k*-NN classifier experiments include: (1) the data without data filtering process experiment, (2) *k*-NN classifier with data filtering process experiment.

4.3.1 *k*-NN Classifier without Data Filtering Process Experiment

In this experiment, the model is built based on the "*k*-NN" classifier developed in a python-based program. Number of nearest neighbors (*k*) is varied and average accuracy is observer. Average accuracy of each number of nearest neighbors (*k*) setting value is shown in Figure 4.16.

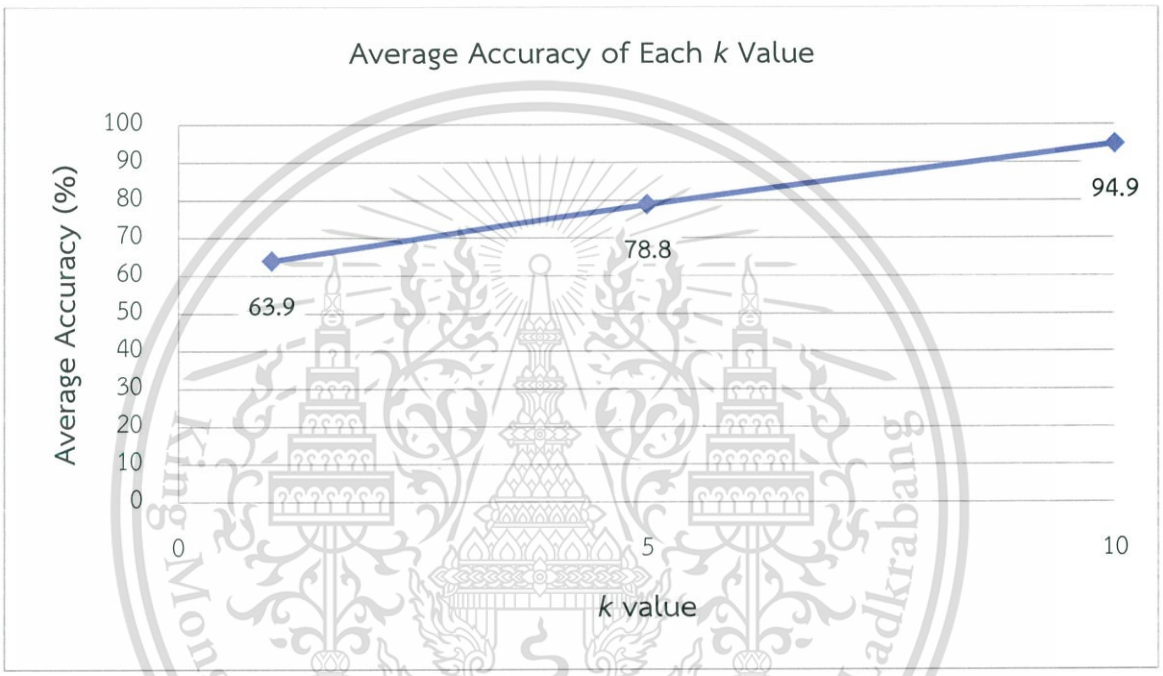

Figure 4.16 Average accuracy of each *k* value (without data filtering process).

The *k* value experiment in Figure 4.16 result shows that *k* value 10 provides the best performance with an average accuracy at 94.9%.

From *k* value experiment overall perspective, the average accuracy increases when *k* value increases.

4.3.2 *k*-NN Classifier with Data Filtering Process Experiment

In this experiment, data filtering process is applied and Number of Nearest Neighbors (*k*) is varied, then average accuracy is observer. Average accuracy of each Number of Nearest Neighbors (*k*) setting value is shown in Figure 4.17.

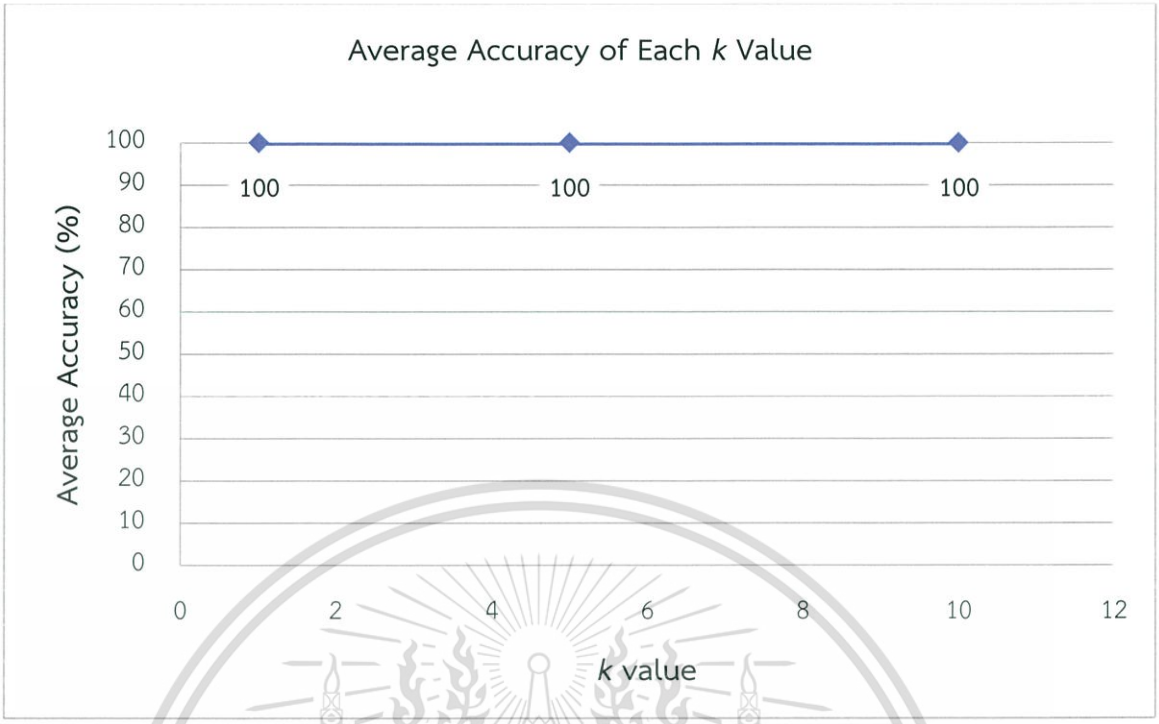

Figure 4.17 Average accuracy of each *k* value (with data filtering process).

Every *k* value provides 100% accuracy. That is means the data filtering process can help improve accuracy significantly. The main reason of improvement from filtering data is it cuts out non-essential data.

4.4 Naïve Bayes Classifier Experiment

The naïve Bayes text classifier has been widely used because of its simplicity in both the training and classifying stages [2-4]. The model is built based on the "Naïve Bayes" classifier developed in RapidMiner 5 [44]. Figure 4.18 illustrates the Naïve Bayes overall setting in RapidMiner 5 used for create prediction model and then measure prediction performance.

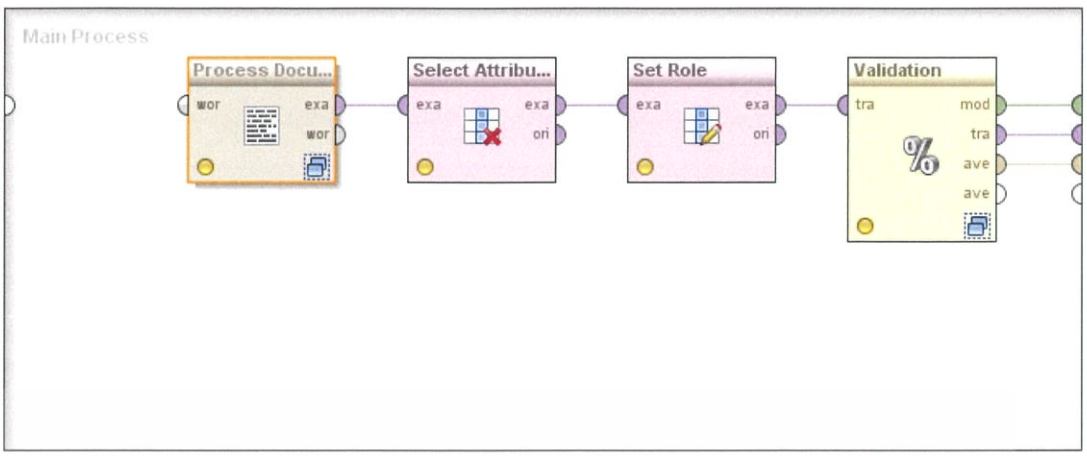

Figure 4.18 The Naïve Bayes overall setting in RapidMiner 5.

In Figure 4.18, Process Document, Select Attributes, Set Role used for convert data in text format to table format which can applied to further process.

Inside Validation operator, it has two subprocesses: a training subprocess and a testing subprocess. The training subprocess is used for training a model. The trained model is then applied in the testing subprocess. The performance of the model is also measured during the testing phase. Figure 4.19 illustrates the validation operator overall setting in Clementine 12.0.

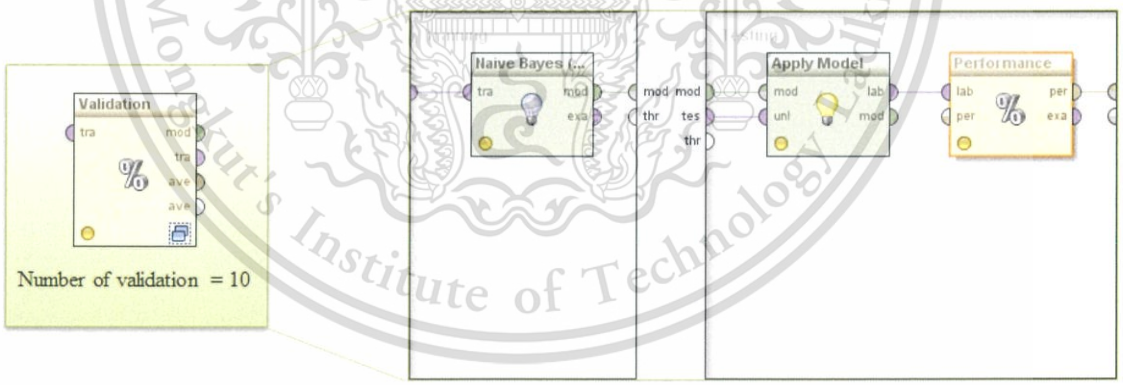

Figure 4.19 The validation operator subprocess setting.

In validation operator, the input data set is partitioned into *k* subsets of equal size. Of the *k* subsets, a single subset is retained as the testing data set (i.e. input of the testing subprocess), and the remaining *k* − 1 subsets are used as training data set (i.e. input of the training subprocess). The cross-validation process is then repeated *k* times,

with each of the *k* subsets used exactly once as the testing data. The *k* results from the *k* iterations then can be averaged (or otherwise combined) to produce a single estimation. In this thesis, the *k* value is set to 10.

In this experiment, data filtering is applied to improve performance of the naïve Bayes classifier. Table 4.1 shows the result of using naïve Bayes classifier to classify the filtered data. It provides an average accuracy at 95.6%.

Table 4.1 Naive Bayes classification results

Model	Average accuracy
Naïve Bayes	95.6 %

CHAPTER 5

CONCLUSION

In this research, an automated prediction failure system has been proposed. Failures can be divided into two categories: false pass and false fail. Four classification algorithms are applied to create a high accuracy model: C5.0, artificial neural network, *k*-NN and the naïve Bayes. The highest average of each of the classification algorithms are shown below.

Table 5.1 False pass drives classification results

Model	Average accuracy
Naïve Bayes	95.6 %
k-NN	100%

Table 5.2 False fail drives classification results

Model	Average accuracy
C5.0	98.4 %
Artificial neural network	92.3 %

Literature on topic 2.9.2 in chapter 2 of this thesis [39] reported that the best accuracy of C5.0 classifier is 93.03%. Better performance with 98.4% accuracy can be achieved in this thesis.

Literature on topic 2.9.3 in chapter 2 of this thesis [8] reported that the best accuracy of artificial neural network classifier is 96.2%, which is better than this thesis achievement 3.9%.

This paper [2] reported that the best accuracy of the naïve Bayes classifier for document classification is 97.0%, which is better than this thesis achievement 1.4%.

This paper [17] reported that the best accuracy of the *k*-NN classifier for document classification is 86.03%. Better performance with 100% accuracy can be achieved in this thesis.

To classify false pass drives, parsing and filtering of the original data was carried out to minimize system resource consumption and improve accuracy by cutting out non-essential data. The filtered data was found to be less than 1% of the total collected data. Moreover, the data filtering process can improve accuracy significantly. The data from *k*-NN classifier show the accuracy without data filtering process provides an average accuracy at 94.9% and the data with data filtering process provides accuracy at 100%.

To make clearer about how *k*-NN classifier get the extremely high accuracy when applying data filtering process, consider Figures 5.1 and 5.2. In both figures, the similarity between false pass drive samples vs. true pass drive samples and false pass drive samples vs. false pass drive samples are plotted in form of standard normal distributions [47]. The normal distribution is parameterized in terms of the mean and the variance. In Figure 5.1, the means of two distributions are very close to each other and they also have some overlap area. As such, it is hard to classify false pass samples correctly. That is the reason why there are still some errors when classified without applying data filtering process.

Next, in Figure 5.2, the means of two distributions are far away from each other and they have no overlap area. As such, it is easy to classify false pass samples. That is the reason why *k*-NN classifier gets the extremely high accuracy when applying data filtering process.

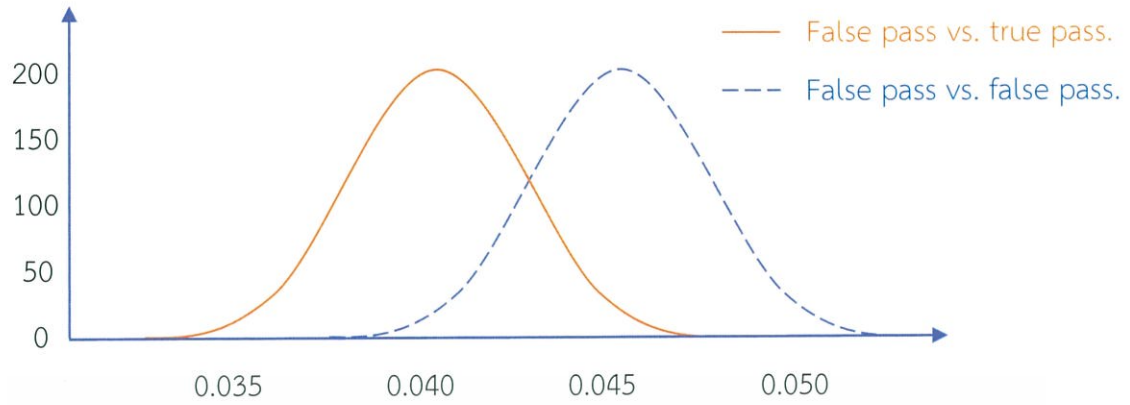

Figure 5.1 Normal distributions of document similarity without data filtering process each class compared with false pass drive class.

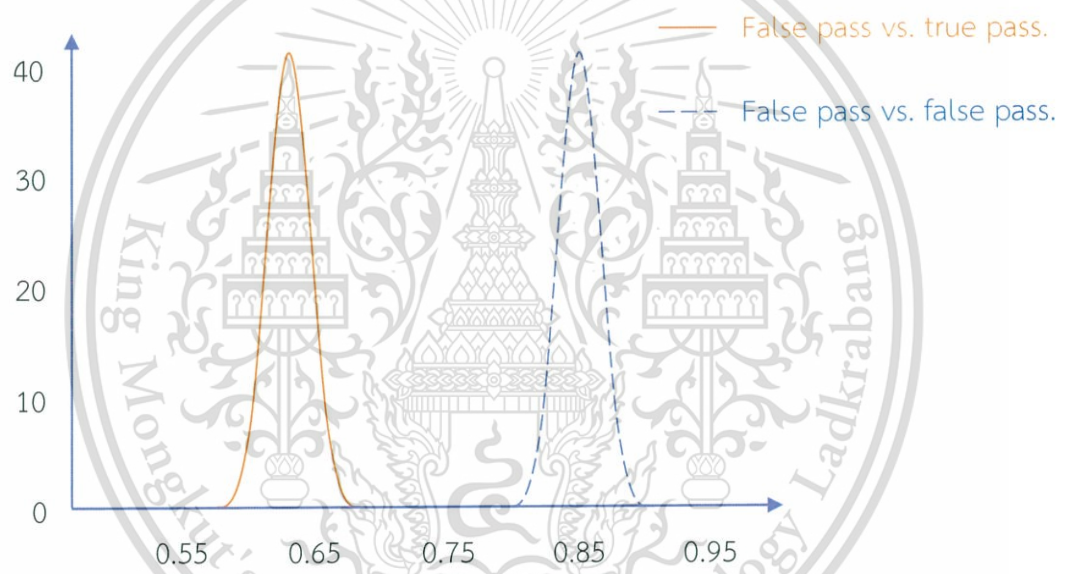

Figure 5.2 Normal distributions of document similarity when applying data filtering process each class compared with false pass drive class.

In Figure 5.1 and 5.2, there is factor for classifying false pass drive samples with *k*-NN classifier. For true pass drive samples consider Figures 5.3 and 5.4. In both figures, the similarity between true pass drive samples vs. false pass drive samples and true pass drive samples vs. true pass drive samples are plotted. In Figure 5.3, the means of two distributions are very close to each other and they also have some overlap area. As such, it is hard to classify true pass samples correctly. That is the reason why there are still some errors when classified without applying data filtering process.

Next, in Figure 5.4, the means of two distributions are far away from each other and they have no overlap area. As such, it is easy to classify false pass samples. That is the reason why *k*-NN classifier gets the extremely high accuracy when applying data filtering process.

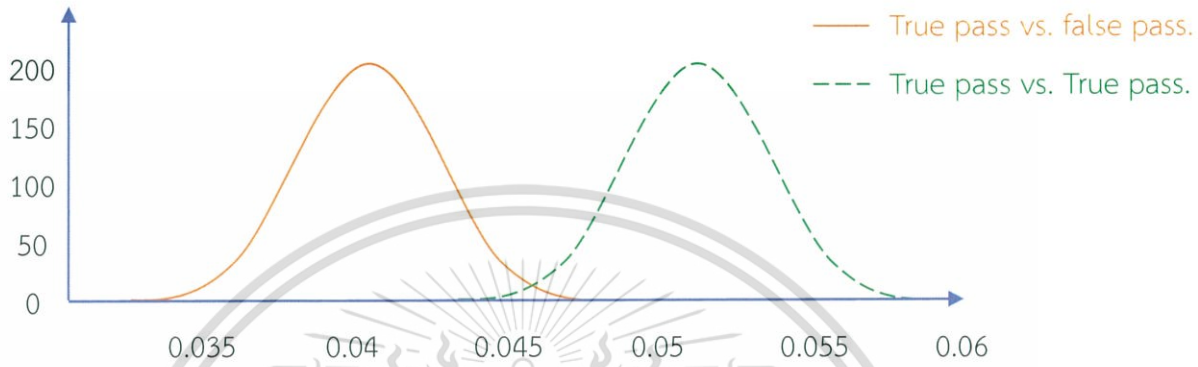

Figure 5.3 Normal distributions of document similarity without data filtering process each class compared with true pass drive class.

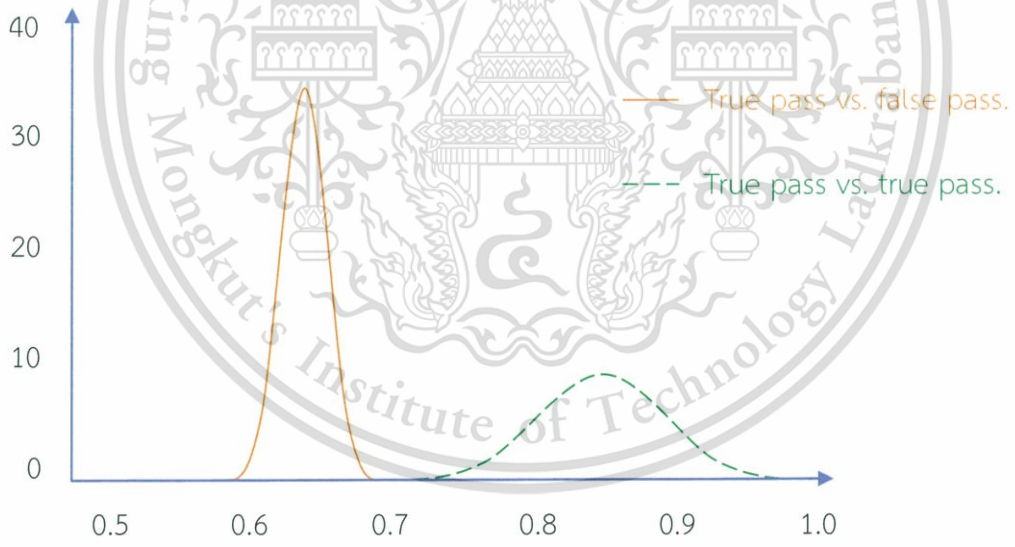

Figure 5.4 Normal distributions of document similarity when applying data filtering process each class compared with true pass drive class.

Regarding the "False Pass" study, it would be possible to take action in terms of classification and process monitoring to avoid potential problems within the test process. It also will help to prevent any incoming problems.

To classify false fail drive, the original data was parsed and filtered down to a set of parameters. The experiment results show that C5.0 classification method provides the best performance with an average accuracy at 98.4%.

To make clearer about how C5.0 classifier get the extremely high accuracy, consider Figure 5.5. In this figure, C5.0 classifier shows the powerful of boosting method that provides extremely high accuracy in our experiment. The idea of boosting method is to generate several classifiers (10 for this thesis) rather than just one. When a new case is to be classified, each classifier votes for its predicted class and the votes are counted to determine the final class [46]. Consider the "Rule 1" in Figure 5.5, each rule represents 1 model, actually the "Rule 1" accuracy is only 90.01% but when combined with other 9 models the combination model (the boosted model) can predict all data correctly. If compare just 1 model of C5.0 classifier without boosting method, the accuracy of C5.0 classifier is lower than artificial neural network classifier. As such, the boosting method is the reason why C5.0 classifier provides best result and overcome artificial neural network classifier.

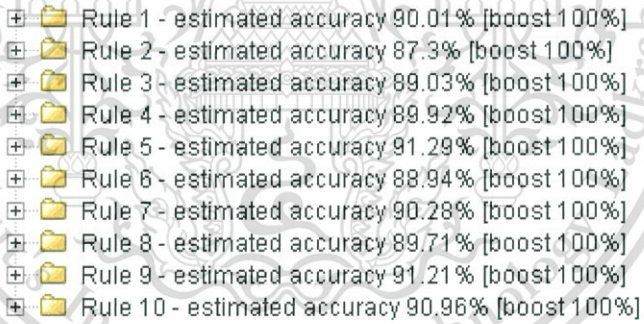

Figure 5.5 An overview of the best model from C5.0 classifier.

Regarding the "False Fail" study, this idea will help us to reduce drive costs dramatically (by reducing field issues and waste costs). Currently, the idea of recovery process is applied in real-world HDD production and it helps the factory reduce costs dramatically.

REFERENCES

[1] Z. Zheng, Z. Lan, Byung H. Park, and A. Geist, "System Log Pre-processing to Improve Failure Prediction," Dependable Systems & Networks 2009.

[2] S. L. Ting, W. H. Ip, Albert H.C. Tsang, "Is Naïve Bayes a Good Classifier for Document Classification?," International Journal of Software Engineering and Its Applications Vol. 5, No. 3, July, 2011.

[3] H. Zhang and J. Su, "Naive Bayesian Classifiers for Ranking," Machine Learning: ECML 2004.

[4] P. Langley and S. Sage, "Induction of Selective Bayesian Classifiers," ISLE 1994.

[5] Y. Freund and L. Mason, "The Alternating Decision Tree Learning Algorithm," Proceedings of the Sixteenth International Conference on Machine Leaning Pages 124-133 ICML 1999.

[6] Y. C. Phen, "Threat analysis using artificial neural network," Faculty of Computer Science and Information System University Technology Malaysia, April 2009.

[7] R. Prasad. P, Nagarjuna D, and A. M. Nejad, "k Nearest Neighbors Algorithm," University of Mysore 2009.

[8] Thanapong T, "Study of Neural Network for Fly Height Failure Pattern Classification in Hard Disk Drive," College of data storage innovation King Mongkut's Institute of Technology Ladkrabang, 2014.

[9] Available: http://talkelab.ucsd.edu/index.php/hdd-intro.

[10] Nie Jianbin, "Control Design and Implementation of Hard Disk Drive Servos," University of California, Berkeley, 2011.

[11] Uwe Boettcheret al., "Servo signal data processing for flying height control in hard disk drives," Microsystem Technologies Journal, vol. 17, Issue 5-7 , pp 937-944, Jun 2012.

[12] Brian D. Strom et al., "Hard Disk Drive Reliability Modeling and Failure Prediction," IEEE transactions on Magnetics, vol. 43, no. 9, Sep 2007.

[13] Jia-Yang Juang et al., "Numerical and Experiment Analyses of Nanometer-Scale Flying Height Control of Magnetic Head with Heating Element," IEEE transactions on Magnetics, vol. 44, no. 11, Nov 2008.

[14] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth "Knowledge Discovery in Databases," American Association for Artificial Intelligence, 1996.

[15] Available: http://web.engr.illinois.edu/~hanj/pdf/ency99.pdf.

[16] Y. Ramamohan, K. Vasantharao, C. Kalyana Chakravarti, and A.S.K.Ratnam , "A Study of Data Mining Tools in Knowledge Discovery Process," International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-2, Issue-3, July 2012.

[17] Y. H. Li, and A. K. Jain, "Classification of Text Document," The computer journal vol. 41, No. 8, 1998.

[18] Tom M. Mitchell, "Machine Learning," McGraw-Hill, NewYork, 1997.

[19] Vangelis Metsis, Ion Androutsopoulos, and Georgios Paliouras, "Spam Filtering with Naive Bayes – Which Naive Bayes?," Third Conference on Email and Anti-Spam (CEAS), July 2006.

[20] Available: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.

[21] H. kou, and G. Gardarin, "Similarity Model and Term Association for Document Categorization," Database and Expert Systems Applications 2002.

[22] Li Baoli, Yu Shiwen, Lu Qin, "An Improved k-Nearest Neighbor Algorithm for Text Categorization," The 20th International Conference on Computer Processing of Oriental Languages, Shenyang, China, 2003.

[23] Manning C. D., Schutze H., "Foundations of Statistical Natural Language Processing," MIT Press, Cambridge, 1999.

[24] Yang Y, Liu X, "A Re-examination of Text Categorization Methods," The 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 42-49, 1999.

[25] Joachims T., "Text Categorization with Support Vector Machines: Learning with Many Relevant Features," The European Conference on Machine Learning, 1998.

[26] Daniel T. Larose, "Discovering Knowledge in Data," A John Wiley & Sons, INC., Publication, 2005.

[27] Vinicius Gonçalves Maltarollo, Káthia Maria Honório, Albérico Borges Ferreira da Silva (2013), "Applications of Artificial Neural Networks in Chemical Problems, Artificial Neural Networks – Architectures and Applications, Prof. Kenji Suzuki (Ed.)," InTech,

DOI: 10.5772/51275. Available from:http://www.intechopen.com/books/artificial-neural-networks-architectures-and-applications/applications-of-artificial-neural-networks-in-chemical-problems.

[28] Available: http://neuralnetworksanddeeplearning.com/chap1.html#sigmoid_neurons.

[29] Available: https://www.metacademy.org/graphs/concepts/perceptron.

[30] Available: http://natureofcode.com/book/chapter-10-neural-networks/.

[31] Hojjat Adeli, and Shih-Lin Huang, "Machine Learning Neural Networks, Genetic Algorithms, and Fuzzy Systems," John Wiley & Sons Ltd., 1995.

[32] Andrej Krenker, Janez Bester, Andrej Kos, "Introduction to the Artificial Neural Networks, Artificial Neural Networks - Methodological Advances and Biomedical Applications, Prof. Kenji Suzuki (Ed.)," ISBN: 978-953-307-243-2, InTech, 2011, Available from: http://www.intechopen.com/books/artificial-neural-networksmethodological-advances-and-biomedical-applications/introduction-to-the-artificial-neural-networks.

[33] Nilima Patil, Rekha Lathi, "Comparison of C5.0 & CART Classification algorithms using pruning technique" International Journal of Engineering Research & Technology (IJERT), Vol. 1 Issue 4, ISSN: 2278-0181, June 2012.

[34] Brijain R Patel, Kushik K Rana, "Survey on Decision Tree Algorithm For Classification" International Journal of Engineering Research & Technology (IJEDR), Volume 2, Issue 1, ISSN: 2321-9939, 2014.

[35] Available: http://scikit-learn.org/stable/modules/tree.html.

[36] XindongWu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, Dan Steinberg, "Top 10 algorithms in data mining" IEEE International Conference on Data Mining (ICDM), December 2006.

[37] Rutvija Pandya, and Jayati Pandya, "C5.0 Algorithm to Improved Decision Tree with Feature Selection and Reduced Error Pruning" International Journal of Computer Applications (0975 – 8887) Volume 117 – No. 16, May 2015.

[38] Tepin W, Kidjaidure Y, "Customer Failure Modes Prediction for Hard Disk Drive using Neural Networks Rank-Level Fusion," The 8th Electrical Engineering /Electronics, Computer, Telecommunications and Information, 2011.

[39] T. Ramangkul, J. Ponsawad, "Hard Disk Drive Failure Mode Prediction from SMART Attribute using Data Mining Method," DSTCON 2011.

[40] K. Thompson, "Regular Expression Search Algorithm," Communications of the ACM 1968.

[41] Charles E. Metz, "Basic principles of ROC analysis," Seminars in Nuclear Medicine, Vol. VIII, No. 4 (October), 1978.

[42] Available: https://en.wikipedia.org/wiki/Accuracy_and_precision.

[43] Available: http://machinelearningmastery.com/how-to-choose-the-right-test-options-when-evaluating-machine-learning-algorithms/.

[44] SPSS Inc "Introduction to Clementine," 2003.

[45] Available: http://www.mathworks.com/help/nnet/ug/improve-neural-network-generalization-and-avoid-overfitting.html.

[46] Huang Huamin, Wu Ge, "Introduce data mining with RapidMiner," Syracuse University, EECS, 2008.

[47] Available: https://en.wikipedia.org/wiki/Probability_density_function

APPENDIX A

PUBLICATION

This work has also been published and presented in The 2014 International Technical Conference on Circuits/Systems, Computer and Communication (ITC-CSCC 2014) in Phuket city, Thailand during July 1-4, 2014.

The 29th International Technical Conference on Circuit/Systems Computers and Communications (ITC-CSCC), Phuket, Thailand, July 1-4, 2014

An Alternative Solution for Hard Disk Drive Classification Process Improvement

Natthakritta Rungtalay
International College
King Mongkut's Institute of Technology Ladkrabang 10520, Thailand.
Email: natthakritta.rungtalay@seagate.com
Chanon Warisarn
College of Data Storage Innovation
King Mongkut's Institute of Technology Ladkrabang 10520, Thailand.
Email: kwchanon@kmitl.ac.th

***Abstract*—Currently, the process to improve test classification accuracy usually requires human abilities. This is a very difficult work for a human due to complexity of test process and need to spend a lot of time and cost. Since there are some benefits to use machine learning instead of human, an alternative solution to improve pass-fail classification process is proposed. In this paper, we introduce the use of machine learning to classify misclassified drives. Our experimental results suggest that it is feasible to determine which classification techniques can predict misclassified drives with up 95% and 70% accuracy, respectively.**

***Keywords*—Classification algorithms; Text classification; Prediction model; Naïve Bayes; Decision tree; C5.0; Neuron network; Cosine similarity; k-NN algorithm.**

I. Introduction

In each step of the test process, a large amount of data is generated so called the testing log. Testing logs are used during the test process to record testing history of individual drives and typically used for failure analysis to diagnose the symptom of drive failure. Failed drives are separated from passed drives by a fail event. Possible results of the drive in test process consist of true pass, true fail, false pass (should have failed), and false fail (should have passed) as shown in Fig. 1. The test process should have only 2 portions: true pass and true fail.

The false fail portion is sometimes called over rejection. It unnecessarily increases costs because this portion should have been classified as passed drives instead of waste that requires treatment and disposal costs. We try to do the simulation if we use the machine learning algorithm to classify instead of our current process to see the machine learning algorithm accuracy.

This paper proposes a method to establish a prediction model for classifying the misclassified drives by using machine learning algorithm. We collected 4,842 drive testing profiles to build a high accuracy prediction model. Within the large amount of data in a testing log, there is some useful information and some not so useful information. A filtering process [1] used to eliminate redundant information and reduce unnecessary data was an important first step to allow meaningful analysis of log data, which also helped reduce space and system requirements. We filtered the log data and processed it to be used in prediction algorithms. We applied a number of prediction algorithms such as Naïve Bayes

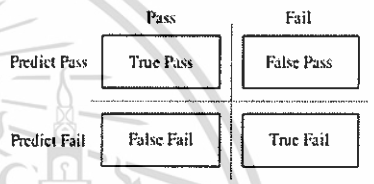

Fig. 1. Possible results in test process.

[2], [3], [4], C5.0 [5], Neuron network [6], and k-NN algorithm [7] which can classify the false fail drives with 100% accuracy.

The paper is organized as follows. After describing our methodology and explaining how to prepare the data with data-preprocessing processes in Section II, we briefly describe the classifier used in this work in Section III. Experimental design and results are given in Sections IV and V, respectively. Finally, Section VI concludes this paper.

II. Methodology

In this work, data is collected from the actual test process in the form of a testing log. The testing log contains information about the drive performance which can be generalized into three basic categories: testing command, testing result, and debug message. The testing command is a low level command sent to the drive to test something (e.g. seek, read, and write). The testing result is the result from the test algorithm performed by the testing command. The debug message is a human readable message for analysis.

Because of differences between false fail and false pass failure symptoms, we have to use 2 different classification methods. Pre-processing depends on the classification method. For the false pass portion, text classification is applied. False pass text classification input data is a text file that containing testing command information. The data for a false fail is a set of some properties of hard disk drives. Therefore, two pre-processing steps will be applied: pre-processing data for the false pass drive portion and pre-processing data for the false fail drive portion.

TABLE I. INTERESTING PROPERTIES FROM THE TESTING LOG.

Description	Type
Failed Head number.	{1, 2, ..., 10}
Temperature upon finding fly height	Continuous
Mean temperature of whole test process.	Continuous
Name of sensor that fail 'Reader or Writer.	{"R","W"}
Count of sensors that fail same inside.	Continuous
Count of retries in finding fly height process.	{1,2,3,4}
Error code of each retry.	Continuous
Contact DAC of each zone.	Continuous

In order to establish a methodology to classify the false pass portion, a testing log was collected from a "Pass" sample at test process. In this paper, we collected 4,142 samples of true pass drives and 21 samples of false pass drives to be used for prediction algorithms. It is hard to find false pass drive samples because it is a kind of event that rarely occurs. In the literature [1], it is shown that filter data can improve the accuracy and effectiveness of classifying algorithms. The filtering process is used to eliminate redundant information and reduce unnecessary data. To reduce unnecessary data, data transformation [3] is applied to convert a set of data values from the data format of a source data system into the data format of a destination data system. The use of regular expressions with arguments [8], a programming technique for locating specific character strings embedded in character text, would allow all instances of a particular pattern to be replaced with another pattern using parts of the original pattern. A longer format of command calling would be replaced with a shorter format using all of the original set of arguments. After filtering, process data will be in a text format and ready for text classification.

In order to establish a methodology to classify the false fail portion, a testing log is collected from a "Fail" sample at test process. In this paper, we select to study drives where the delta fly height between the outer zone and inner zone is over a failure limit, because these failures have a high recovery rate. That means we can get more good drives out of this portion. In the fly height measurement process, the spacing distance between recording head and media are considered in terms of the voltage values that are applied into the heater element of the recording head. It causes the writer and reader elements to protrude until they are in contact with the media, which is detected by the servo detector system. The heater voltage values are in the Digital to Analog Converter (DAC) unit for both read and write operations in each location (data zone) on the media. Data zones are separated into 3 groups: outer disc (OD), middle disc (MD), and inner disc (ID). To build the prediction model, 305 true fail and 374 false fail samples were collected. We then selected interesting properties from testing log to apply to the classification algorithm. See Table I.

III. CLASSIFIERS

In the following sections, we briefly describe the Naïve Bayes classifier, k-NN, Neural Network, and C5.0 classification methods used in our study.

A. Naïve Bayes Classifier

The Naïve Bayes classifier algorithm: Let $C = (c_1, ..., c_m)$ be m document classes. Given a new unlabeled document D and its corresponding word-list $\overline{W} = (w_1, ...w_{d'})$, the Naïve Bayes approach assigns D to a class c^*_{NB} as follows:

$$c^*_{NB} = \arg\max_{c_j \in C} P(c_j) \prod_{i=1}^{d'} P(w_i|c_j), \quad (1)$$

where $P(c_j)$ is the *a priori* probability of class c_j and $P(w_i|c_j)$ is the conditional probability of word w_i given class c_j. The underlying assumption of the Naïve Bayes approach is that for a given class c_j the probabilities of words occurring in a document are independent of each other. When the size of the training set is small, the relative frequency estimates of probabilities, $P(w_i|c_j)$, will not be reasonable; if a word never appears in the given training data, its relative frequency estimate will be zero. Instead, we applied the Laplace law of succession [10] to estimate $P(w_i|c_j)$. The estimate of the probability $P(w_i|c_j)$ is given as:

$$P(w_i|c_j) = \frac{n_{ij} + 1}{n_j + k_j}, \quad (2)$$

where n_j is the total number of words in class c_j, n_{ij} is the number of occurrences of word w_i in class c_j and k_j is the vocabulary size of class c_j. This is the result of the Bayesian estimation with a uniform prior assumption, i.e. probabilities of the occurrence of words appearing in class c_j are equally likely.

B. k-Nearest Neighbors Classifier

For k-Nearest Neighbors or k-NN classifier [6], the training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples. In the classification phase, k is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point. In this work, we use the TF-IDF (TF is the term frequency in a document and IDF is the inverse document frequency) weighting scheme and use the cosine similarity [11], [12] instead of Euclidean distance to measure the similarity of the two documents. Given two documents D_1 and D_2, their corresponding weighted feature vectors are $T_1 = (t_1, \delta_1)_{i=1}^{d}$ and $T_2 = (t_2, \delta_2)_{i=1}^{d}$, where δ_{ki} is the weight of word w_i in document k (TF-IDF). The similarity between D_1 and D_2 is then defined as:

$$S(D_1, D_2) = \frac{T_1^T T_2}{\|T_1\| \, \|T_2\|}, \quad (3)$$

where $\|\cdot\|$ denotes the norm of the vector. T is transpose operator. To save system resources we use k = 1. If k = 1, then the object is simply assigned to the class of its single nearest neighbors.

C. Neural Network Classifier

Neural network is a popular model used in pattern recognition. The basic structure of the neural network consists of input layer, hidden layer, and output layer.

An overview of the Neural network algorithm is shown in Fig. 2, and it includes two calculations. The first is the forward calculation and the second is the error back calculation. In the

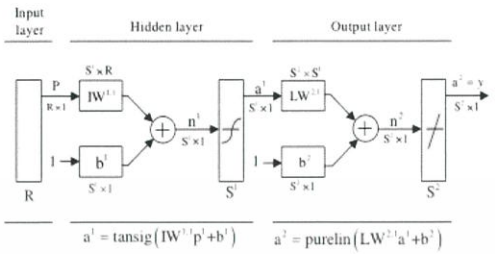

Fig. 2. The model of Neural network.

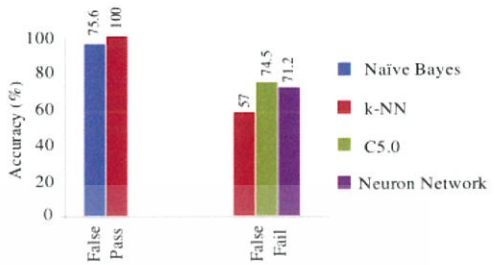

Fig. 3. Classification accuracy in each incorrect prediction portion.

process of forward calculation, input data from the input layer passes through the hidden layer processing and is transmitted to the output layer. If output layer gets an incorrect output result from the target, it will be transferred via back propagation and the weights of each neuron will be recalculated. After correcting the weights of neurons and the network's output becomes consistent with the transfer form again, the actual output and expected error will lead to a new weight correction with the minimum error [7], [10], [13].

D. C5.0 Decision Tree Classifier

The process of a decision tree is to find the best attribute variables of classification ability. Then, through the best attribute variables, the data will be divided into multiple subsets. Within each subset, the best property of the classification capability is found, which can then be used to divide the current subset into multiple subsets further. This is always an iterative operation until all subsets contain only one category or the number of samples is smaller than a certain threshold [5]. Information gain ratio is often used to select properties, and the information gain is:

$$split\_Info(X) = -\sum_{i=1}^{n} \frac{|s_i|}{|s|} \times \log_2\left(\frac{|s_i|}{|s|}\right), \quad (4)$$

$$gain\_ratio(X) = \frac{gain(X)}{split\_Info(X)}, \quad (5)$$

where, $gain(X)$ represents the information gain rooted by X; $split\_Info(X)$ represents potential information generated by dividing optimal threshold s into n parts, s_i is the probability of class i.

IV. Experimental Design

The data that is passed into the pre-processing process will go though some classification such as Naïve Bayes, k-NN, C5.0, and Neuron Network. For the false pass portion, empirical data was filtered down to a primary testing command (the read/write command) to which we can apply machine-learning based algorithms. Because there are a lot of parameters in a passing drive, selecting a small set of parameters is a limitation. To have a stronger model, we need to add all test parameters to the prediction model. There are clear benefits to using text classification, which can keep all parameters and can be applied to machine-learning based algorithms. For false fail drive portion, empirical data was filtered to a set of related parameters, and then machine-learning based algorithms were applied.

TABLE II. False Pass Drives Classification Results.

Model	Accuracy (Testing)
Naïve Bayes	95.56%
k-NN	100%

V. Results and Discussion

The total samples were split into a random 70% training set and a 30% testing set. The prediction accuracy of each classification model is shown in Table II and Table III. Fig. 3 shows the summary of classification accuracy in each incorrect prediction portion. For false pass drives prediction, the model from k-NN algorithm shows the highest accuracy, which is 100%. For Naïve Bayes algorithm, which is classic model for text classification, the model shows 95.56% accuracy. The k-NN algorithm is easier to apply within the actual test process when compared with Naïve Bayes. For false fail drives prediction, the model from C5.0 algorithm shows the highest accuracy, which is 74.49%. Accuracy from C5.0 is not significantly different when compared with Neuron Network but the model from C5.0 is easier to apply within the actual test process. It is also easier to understand by humans. As such, there are clear benefits to using the C5.0 algorithm.

VI. Conclusions

An automated prediction failure system has been proposed. For false pass drive classification, parsing and filtering of the original data was carried out to minimize system resource consumption and improve accuracy by cutting out non-essential data. The filtered data was found to be less than 1% of the total collected data. For false fail drive classification, the original data was parsed and filtered down to a set of parameters. The experiment results show that the k-NN and C5.0 classification methods were promising for predicting incorrectly classified drives. Using the results of this study it would be possible to take action in terms of classification and process monitoring to avoid potential problems within the test process. This idea will help us to reduce drive costs dramatically (by reducing

TABLE III. FALSE FAIL DRIVES CLASSIFICATION RESULTS.

Model	Accuracy (Testing)
C5.0.	74.49%
Neuron Network.	71.19%
k-NN.	57.00%

field issues and waste costs). It also will help to prevent any incoming problems.

ACKNOWLEDGMENT

This work was supported by College of Data Storage Innovation (D*STAR). King Mongkut's Institute of Technology Ladkrabang Research Fund, Thailand.

REFERENCES

[1] Z. Zheng, Z. Lan, B. H. Park, and A. Geist, "System Log Pre-processing to Improve Failure Prediction," Dependable Systems & Networks 2009.

[2] S. L. Tang, W. H. Ip, and A. H. C. Tsang, "Is Naïve Bayes a Good Classifier for Document Classification?," International Journal of Software Engineering and Its Applications Vol. 5, No. 3, July, 2011.

[3] H. Zhang and J. Su, "Naïve Bayesian Classifiers for Ranking," Machine Learning: ECML 2004.

[4] P. Langley and S. Sage, "Induction of Selective Bayesian Classifiers," ISLE 1994.

[5] Y. Freund and L. Mason, "The Alternating Decision Tree Learning Algorithm," Proceedings of the Sixteenth International Conference on Machine Leaning Pages 124-133 ICML 1999.

[6] Y. C. Phen, "Threat analysis using artificial neural network," Faculty of Computer Science and Information System University Technology Malaysia, April 2009.

[7] R. Prasad, D. Nagarjuna, and A. M. Nejad, "k Nearest Neighbors Algorithm," University of Mysore 2009.

[8] K. Thompson, "Regular Expression Search Algorithm," Communications of the ACM 1968.

[9] J. K. Basu, D. Bhattacharyya, and T. Kim, "Use of Artificial Neural Network in Pattern Recognition," International Journal of Software Engineering and Its Applications Vol. 4, No. 2, April 2010.

[10] F. Yan, W. Liu, and L. Tian, "A New Neural Network Approach for Fault Location of Distribution Network," International Conference on Mechatronic Science, Electric Engineering and Computer (MEC 2011), August 2011.

[11] H. kou and G. Gardarin, "Similarity Model and Term Association for Document Categorization," Database and Expert Systems Applications 2002.

[12] Y. H. Li and A. K. Jain, "Classification of Text Document," The computer journal vol. 41, No. 8, 1998.

[13] X. Ren and X. Hou, "The Misalignment Fault Model Building for Rotating Machinery Rotor Based on BP Network," 8th International Conference on Natural Computation (ICNC 2012), May 2012.

APPENDIX B

PYTHON SOURCE CODE

In this thesis, "*k*-NN" classifier was developed on python-based program. The source code consists of 3 files that are logFileSimilarity.py, validation_text_similarity.py, and porter.py.

logFileSimilarity.py

Propose of this file is compare of each two documents. To compare documents, Cosine similarity is applied. Cosine similarity is a measure of similarity between two vectors of an inner product space that measures the cosine of the angle between them. The consine of 0° is 1, two vector at 90° have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude. The source code as shown below.

```
1.  import re
2.  import porter
3.  from numpy import zeros,dot
4.  from numpy.linalg import norm
5.  import os
6.
7.  __all__=['compare']
8.
9.  # import real stop words
10. #stop_words = [ 'i', 'in', 'a', 'to', 'the', 'it', 'have', 'haven\'t', 'was', 'b
    ut', 'is', 'be', 'from' ]
11. stop_words = [w.strip() for w in open('english.stop','r').readlines()]
12. print stop_words
13.
14. splitter=re.compile ( "[a-z\-']+", re.I )
15. stemmer=porter.PorterStemmer()
16.
17. def add_word(word,d,do_stem = 0):
18.     """
19.         Adds a word the a dictionary for words/count
20.         first checks for stop words
21.       the converts word to stemmed version
22.     """
23.     w=word.lower()
24.     if w not in stop_words:
25.         if do_stem:
26.             ws=stemmer.stem(w,0,len(w)-1)
27.         else:
28.             ws=w
29.         d.setdefault(ws,0)
```

```
30.       d[ws] += 1
31.
32. def doc_vec(line_of_doc,key_idx,do_stem = 0):
33.    v = zeros(len(key_idx))
34.    for line in line_of_doc:
35.       if line.strip() == "":
36.         continue
37.
38.       if do_stem:
39.          keydata = key_idx.get(stemmer.stem(line,0,len(line)-1).lower(), None)
40.       else:
41.          keydata = key_idx.get(line.strip().lower(), None)
42.       if keydata:
43.          v[keydata[0]] = 1
44.    return v
45.
46. def compare(doc1,doc2):
47.
48.    # strip all punctuation but - and '
49.    # convert to lower case
50.    # store word/occurance in dict
51.    all_words=dict()
52.
53.    f = open(doc1,'r')
54.    lines_in_doc1 = f.readlines()
55.    f.close()
56.
57.    f = open(doc2,'r')
58.    lines_in_doc2 = f.readlines()
59.    f.close()
60.
61.    for line in lines_in_doc1 + lines_in_doc2:
62.       if line.strip() != "":
63.          add_word(line.strip(),all_words)
64.
65.    # build an index of keys so that we know the word positions for the vector
66.    key_idx=dict() # key-> ( position, count )
67.    keys=all_words.keys()
68.    keys.sort()
69.    #print keys
70.    for i in range(len(keys)):
71.       key_idx[keys[i]] = (i,all_words[keys[i]])
72.    del keys
73.    del all_words
74.
75.    v1=doc_vec(lines_in_doc1,key_idx)
76.    v2=doc_vec(lines_in_doc2,key_idx)
77.    return float(dot(v1,v2) / (norm(v1) * norm(v2)))
78.
79.
80. if __name__ == '__main__':
81.    doc1 = "D:\\Learning\\MasterDegree\\Thesis\\__CodeVerThesis\\LB_Drive_Escape\
    \FIN2_PASS_FAKE\\LEAN_LOG\\OP2.030.Z3002V43.FIN2.txt.log"
82.    doc2 = "D:\\Learning\\MasterDegree\\Thesis\\__CodeVerThesis\\LB_Drive_Escape\
    \FIN2_PASS_FAKE\\LEAN_LOG\\OP3.042.Z3003XSG.FIN2.txt.log"
83.    #doc2 = "D:\\Learning\\MasterDegree\\Thesis\\__CodeVerThesis\\LB_Drive_Escape
    \\FIN2_PASS_TRUE\\LEAN_LOG\\OP2.041.Z3002EH4.FIN2.txt.log"
84.    print "Using Doc1: %s\n\nUsing Doc2: %s\n" % ( doc1, doc2 )
85.
86.    print "Similarity %s" % compare(doc1,doc2)
```

validation_text_similarity.py

Propose of this file is finding accuracy of k-NN model. The validation process will compare result from text-classification with human classification. If result from text-classification equal to human classification, result will justify as correct. If result from text-classification not equal to human classification, result will justify as incorrect. The source code as shown below.

```
1.  import os,traceback
2.  import numpy
3.
4.  from logFile_similarity import compare
5.
6.
7.  current_dir = os.path.dirname(__file__)
8.  all_testing_class_dir  = os.path.join(current_dir,'TESTING_CLASS_LEAN')
9.  all_training_class_dir = os.path.join(current_dir,'TRAINING_CLASS_LEAN')
10.
11. class_similarity_score = []
12. class_name_list        = []
13.
14. test_name_list = []
15. testing_result_details = {}
16.
17. confusion_matrix = {}
18.
19. for testing_class_name in os.listdir(all_testing_class_dir):
20.
21.    testingClassDir = os.path.join(all_testing_class_dir,testing_class_name)
22.    for test_file in os.listdir(testingClassDir):
23.       trainingClassList = []
24.       testFilePath = os.path.join(testingClassDir,test_file)
25.       test_name_list.append(test_file)
26.       testing_result_details.update({test_file:{ "HUMAN_CLASSIFY_AS" : testing_c
    lass_name }})
27.
28.       for training_class_name in os.listdir(all_training_class_dir):
29.          trainingClassList.append(training_class_name)
30.          trainingClassDir = os.path.join(all_training_class_dir,training_class_n
    ame)
31.
32.          testing_result_details[test_file].update( { training_class_name:{} } )

33.          testing_result_details[test_file][training_class_name].update( { "simil
    arity_details":{} } )
34.          testing_result_details[test_file][training_class_name].update( { "train
    ing_file_name_list":[] } )
35.          testing_result_details[test_file][training_class_name].update( { "train
    ing_file_score_list":[] } )
36.          for train_file in os.listdir(trainingClassDir):
37.             trainFilePath = os.path.join(trainingClassDir,train_file)
38.             try:
39.               percent_similarity = compare(testFilePath,trainFilePath)
```

```
40.             except:
41.                print traceback.format_exc()
42.             print "similarity %s VS %s = %s" %(test_file,train_file,percent_simi
    larity)
43.             testing_result_details[test_file][training_class_name]["training_fil
    e_name_list"].append(train_file)
44.             testing_result_details[test_file][training_class_name]["training_fil
    e_score_list"].append(percent_similarity)
45.             testing_result_details[test_file][training_class_name]["similarity_d
    etails"].update({train_file:percent_similarity})
46.          testing_result_details[test_file]["class_name_list"] = []
47.          testing_result_details[test_file]["class_score_list"] = []
48.          testing_result_details[test_file]["class_score_details"] = {}
49.       for class_name in trainingClassList:
50.          this_class_score = int(numpy.mean(testing_result_details[test_file][cla
    ss_name]["training_file_score_list"]) * 100)
51.          testing_result_details[test_file]["class_name_list"].append(class_name)
52.          testing_result_details[test_file]["class_score_list"].append(this_class
    _score)
53.          testing_result_details[test_file]["class_score_details"].update( {class
    _name:this_class_score} )
54.
55.       maximum_likelihood_score          = max(testing_result_details[test_file]
    ["class_score_list"])
56.       maximum_likelihood_reference_index = testing_result_details[test_file]["cl
    ass_score_list"].index(maximum_likelihood_score)
57.       maximum_likelihood_class_name     = testing_result_details[test_file]["cl
    ass_name_list"][maximum_likelihood_reference_index]
58.       testing_result_details[test_file]["MODEL_CLASSIFY_AS"] = maximum_likelihoo
    d_class_name
59.       if not confusion_matrix.has_key("%s_%s" %(testing_class_name,maximum_likel
    ihood_class_name) ):
60.          confusion_matrix[ "%s_%s" %( testing_class_name,maximum_likelihood_clas
    s_name) ] = 0
61.       confusion_matrix[ "%s_%s" %( testing_class_name,maximum_likelihood_class_n
    ame ) ] += 1
62.
63. print confusion_matrix
64. f = open("D:\\Learning\\MasterDegree\\Thesis\\__CodeVerThesis\\similarity\\RESUL
    T_TEST_FULL_LOG.txt",'w')
65. f.write(repr(confusion_matrix))
66. f.close()
```

porter.py

This file contain the Porter stemming algorithm is a process for removing suffixes from words in English, Removing suffixed automatically is an operation which is especially useful in the field of information retrieval. Normally, a document is represented by a vector of words. Word with a common stem will usually have similar meanings, for example:

Connect

Connected

Connecting

Connection

Connections

This may be done by removal of the various suffixes –ED, -ING, -ION, IONS. In addition, the suffix stripping process will reduce the total number of terms in the system. The source code of this process as shown below.

```
#!/usr/bin/env python

"""Porter Stemming Algorithm
This is the Porter stemming algorithm, ported to Python from the
version coded up in ANSI C by the author. It may be be regarded
as canonical, in that it follows the algorithm presented in

Porter, 1980, An algorithm for suffix stripping, Program, Vol. 14,
no. 3, pp 130-137,

only differing from it at the points maked --DEPARTURE-- below.

See also http://www.tartarus.org/~martin/PorterStemmer

The algorithm as described in the paper could be exactly replicated
by adjusting the points of DEPARTURE, but this is barely necessary,
because (a) the points of DEPARTURE are definitely improvements, and
(b) no encoding of the Porter stemmer I have seen is anything like
as exact as this version, even with the points of DEPARTURE!

Vivake Gupta (v@nano.com)

Release 1: January 2001

Further adjustments by Santiago Bruno (bananabruno@gmail.com)
to allow word input not restricted to one word per line, leading
to:

release 2: July 2008
"""

import sys
```

```
33.
34. class PorterStemmer:
35.
36.     def __init__(self):
37.         """The main part of the stemming algorithm starts here.
38.         b is a buffer holding a word to be stemmed. The letters are in b[k0],
39.         b[k0+1] ... ending at b[k]. In fact k0 = 0 in this demo program. k is
40.         readjusted downwards as the stemming progresses. Zero termination is
41.         not in fact used in the algorithm.
42.
43.         Note that only lower case sequences are stemmed. Forcing to lower case
44.         should be done before stem(...) is called.
45.         """
46.
47.         self.b = ""  # buffer for word to be stemmed
48.         self.k = 0
49.         self.k0 = 0
50.         self.j = 0   # j is a general offset into the string
51.
52.     def cons(self, i):
53.         """cons(i) is TRUE <=> b[i] is a consonant."""
54.         if self.b[i] == 'a' or self.b[i] == 'e' or self.b[i] == 'i' or self.b[i]
    == 'o' or self.b[i] == 'u':
55.             return 0
56.         if self.b[i] == 'y':
57.             if i == self.k0:
58.                 return 1
59.             else:
60.                 return (not self.cons(i - 1))
61.         return 1
62.
63.     def m(self):
64.         """m() measures the number of consonant sequences between k0 and j.
65.         if c is a consonant sequence and v a vowel sequence, and <..>
66.         indicates arbitrary presence,
67.
68.             <c><v>       gives 0
69.             <c>vc<v>     gives 1
70.             <c>vcvc<v>   gives 2
71.             <c>vcvcvc<v> gives 3
72.             ....
73.         """
74.         n = 0
75.         i = self.k0
76.         while 1:
77.             if i > self.j:
78.                 return n
79.             if not self.cons(i):
80.                 break
81.             i = i + 1
82.         i = i + 1
83.         while 1:
84.             while 1:
85.                 if i > self.j:
86.                     return n
87.                 if self.cons(i):
88.                     break
89.                 i = i + 1
90.             i = i + 1
91.             n = n + 1
92.             while 1:
```

```
93.                 if i > self.j:
94.                     return n
95.                 if not self.cons(i):
96.                     break
97.                 i = i + 1
98.             i = i + 1
99.
100.            def vowelinstem(self):
101.                """vowelinstem() is TRUE <=> k0,...j contains a vowel"""
102.                for i in range(self.k0, self.j + 1):
103.                    if not self.cons(i):
104.                        return 1
105.                return 0
106.
107.            def doublec(self, j):
108.                """doublec(j) is TRUE <=> j,(j-
    1) contain a double consonant."""
109.                if j < (self.k0 + 1):
110.                    return 0
111.                if (self.b[j] != self.b[j-1]):
112.                    return 0
113.                return self.cons(j)
114.
115.            def cvc(self, i):
116.                """cvc(i) is TRUE <=> i-2,i-1,i has the form consonant - vowel -
     consonant
117.                and also if the second c is not w,x or y. this is used when tryi
    ng to
118.                restore an e at the end of a short  e.g.
119.
120.                    cav(e), lov(e), hop(e), crim(e), but
121.                    snow, box, tray.
122.                """
123.                if i < (self.k0 + 2) or not self.cons(i) or self.cons(i-
    1) or not self.cons(i-2):
124.                    return 0
125.                ch = self.b[i]
126.                if ch == 'w' or ch == 'x' or ch == 'y':
127.                    return 0
128.                return 1
129.
130.            def ends(self, s):
131.                """ends(s) is TRUE <=> k0,...k ends with the string s."""
132.                length = len(s)
133.                if s[length - 1] != self.b[self.k]: # tiny speed-up
134.                    return 0
135.                if length > (self.k - self.k0 + 1):
136.                    return 0
137.                if self.b[self.k-length+1:self.k+1] != s:
138.                    return 0
139.                self.j = self.k - length
140.                return 1
141.
142.            def setto(self, s):
143.                """setto(s) sets (j+1),...k to the characters in the string s, r
    eadjusting k."""
144.                length = len(s)
145.                self.b = self.b[:self.j+1] + s + self.b[self.j+length+1:]
146.                self.k = self.j + length
147.
148.            def r(self, s):
```

```
149.                """r(s) is used further down."""
150.                if self.m() > 0:
151.                    self.setto(s)
152.
153.            def step1ab(self):
154.                """step1ab() gets rid of plurals and -ed or -ing. e.g.
155.
156.                   caresses  ->  caress
157.                   ponies    ->  poni
158.                   ties      ->  ti
159.                   caress    ->  caress
160.                   cats      ->  cat
161.
162.                   feed      ->  feed
163.                   agreed    ->  agree
164.                   disabled  ->  disable
165.
166.                   matting   ->  mat
167.                   mating    ->  mate
168.                   meeting   ->  meet
169.                   milling   ->  mill
170.                   messing   ->  mess
171.
172.                   meetings  ->  meet
173.                """
174.                if self.b[self.k] == 's':
175.                    if self.ends("sses"):
176.                        self.k = self.k - 2
177.                    elif self.ends("ies"):
178.                        self.setto("i")
179.                    elif self.b[self.k - 1] != 's':
180.                        self.k = self.k - 1
181.                if self.ends("eed"):
182.                    if self.m() > 0:
183.                        self.k = self.k - 1
184.                elif (self.ends("ed") or self.ends("ing")) and self.vowelinstem(
    ):
185.                    self.k = self.j
186.                    if self.ends("at"):   self.setto("ate")
187.                    elif self.ends("bl"): self.setto("ble")
188.                    elif self.ends("iz"): self.setto("ize")
189.                    elif self.doublec(self.k):
190.                        self.k = self.k - 1
191.                        ch = self.b[self.k]
192.                        if ch == 'l' or ch == 's' or ch == 'z':
193.                            self.k = self.k + 1
194.                    elif (self.m() == 1 and self.cvc(self.k)):
195.                        self.setto("e")
196.
197.            def step1c(self):
198.                """step1c() turns terminal y to i when there is another vowel in
    the stem."""
199.                if (self.ends("y") and self.vowelinstem()):
200.                    self.b = self.b[:self.k] + 'i' + self.b[self.k+1:]
201.
202.            def step2(self):
203.                """step2() maps double suffices to single ones.
204.                so -ization ( = -ize plus -ation) maps to -
    ize etc. note that the
205.                string before the suffix must give m() > 0.
206.                """
```

```
207.                if self.b[self.k - 1] == 'a':
208.                    if self.ends("ational"):   self.r("ate")
209.                    elif self.ends("tional"):  self.r("tion")
210.                elif self.b[self.k - 1] == 'c':
211.                    if self.ends("enci"):      self.r("ence")
212.                    elif self.ends("anci"):    self.r("ance")
213.                elif self.b[self.k - 1] == 'e':
214.                    if self.ends("izer"):      self.r("ize")
215.                elif self.b[self.k - 1] == 'l':
216.                    if self.ends("bli"):       self.r("ble") # --DEPARTURE--
217.                    # To match the published algorithm, replace this phrase with
                    #   if self.ends("abli"):      self.r("able")
218.                    elif self.ends("alli"):    self.r("al")
219.                    elif self.ends("entli"):   self.r("ent")
220.                    elif self.ends("eli"):     self.r("e")
221.                    elif self.ends("ousli"):   self.r("ous")
222.                elif self.b[self.k - 1] == 'o':
223.                    if self.ends("ization"):   self.r("ize")
224.                    elif self.ends("ation"):   self.r("ate")
225.                    elif self.ends("ator"):    self.r("ate")
226.                elif self.b[self.k - 1] == 's':
227.                    if self.ends("alism"):     self.r("al")
228.                    elif self.ends("iveness"): self.r("ive")
229.                    elif self.ends("fulness"): self.r("ful")
230.                    elif self.ends("ousness"): self.r("ous")
231.                elif self.b[self.k - 1] == 't':
232.                    if self.ends("aliti"):     self.r("al")
233.                    elif self.ends("iviti"):   self.r("ive")
234.                    elif self.ends("biliti"):  self.r("ble")
235.                elif self.b[self.k - 1] == 'g': # --DEPARTURE--
236.                    if self.ends("logi"):      self.r("log")
237.                # To match the published algorithm, delete this phrase
238.
239.            def step3(self):
240.                """step3() dels with -ic-, -full, -
241.    ness etc. similar strategy to step2."""
242.                if self.b[self.k] == 'e':
243.                    if self.ends("icate"):     self.r("ic")
244.                    elif self.ends("ative"):   self.r("")
245.                    elif self.ends("alize"):   self.r("al")
246.                elif self.b[self.k] == 'i':
247.                    if self.ends("iciti"):     self.r("ic")
248.                elif self.b[self.k] == 'l':
249.                    if self.ends("ical"):      self.r("ic")
250.                    elif self.ends("ful"):     self.r("")
251.                elif self.b[self.k] == 's':
252.                    if self.ends("ness"):      self.r("")
253.
254.            def step4(self):
255.                """step4() takes off -ant, -
256.    ence etc., in context <c>vcvc<v>."""
257.                if self.b[self.k - 1] == 'a':
258.                    if self.ends("al"): pass
259.                    else: return
260.                elif self.b[self.k - 1] == 'c':
261.                    if self.ends("ance"): pass
262.                    elif self.ends("ence"): pass
263.                    else: return
264.                elif self.b[self.k - 1] == 'e':
                    if self.ends("er"): pass
```

```
265.                    else: return
266.                elif self.b[self.k - 1] == 'i':
267.                    if self.ends("ic"): pass
268.                    else: return
269.                elif self.b[self.k - 1] == 'l':
270.                    if self.ends("able"): pass
271.                    elif self.ends("ible"): pass
272.                    else: return
273.                elif self.b[self.k - 1] == 'n':
274.                    if self.ends("ant"): pass
275.                    elif self.ends("ement"): pass
276.                    elif self.ends("ment"): pass
277.                    elif self.ends("ent"): pass
278.                    else: return
279.                elif self.b[self.k - 1] == 'o':
280.                    if self.ends("ion") and (self.b[self.j] == 's' or self.b[sel
    f.j] == 't'): pass
281.                    elif self.ends("ou"): pass
282.                    # takes care of -ous
283.                    else: return
284.                elif self.b[self.k - 1] == 's':
285.                    if self.ends("ism"): pass
286.                    else: return
287.                elif self.b[self.k - 1] == 't':
288.                    if self.ends("ate"): pass
289.                    elif self.ends("iti"): pass
290.                    else: return
291.                elif self.b[self.k - 1] == 'u':
292.                    if self.ends("ous"): pass
293.                    else: return
294.                elif self.b[self.k - 1] == 'v':
295.                    if self.ends("ive"): pass
296.                    else: return
297.                elif self.b[self.k - 1] == 'z':
298.                    if self.ends("ize"): pass
299.                    else: return
300.                else:
301.                    return
302.                if self.m() > 1:
303.                    self.k = self.j
304.
305.            def step5(self):
306.                """step5() removes a final -e if m() > 1, and changes -ll to -
    l if
307.                m() > 1.
308.                """
309.                self.j = self.k
310.                if self.b[self.k] == 'e':
311.                    a = self.m()
312.                    if a > 1 or (a == 1 and not self.cvc(self.k-1)):
313.                        self.k = self.k - 1
314.                if self.b[self.k] == 'l' and self.doublec(self.k) and self.m() >
    1:
315.                    self.k = self.k -1
316.
317.            def stem(self, p, i, j):
318.                """In stem(p,i,j), p is a char pointer, and the string to be ste
    mmed
319.                is from p[i] to p[j] inclusive. Typically i is zero and j is the
320.                offset to the last character of a string, (p[j+1] == '\0'). The
```

```
321.                stemmer adjusts the characters p[i] ... p[j] and returns the new

322.                end-
   point of the string, k. Stemming never increases word length, so
323.                i <= k <= j. To turn the stemmer into a module, declare 'stem' a
   s
324.                extern, and delete the remainder of this file.
325.                """
326.                # copy the parameters into statics
327.                self.b = p
328.                self.k = j
329.                self.k0 = i
330.                if self.k <= self.k0 + 1:
331.                    return self.b # --DEPARTURE--
332.
333.                # With this line, strings of length 1 or 2 don't go through the
334.                # stemming process, although no mention is made of this in the
335.                # published algorithm. Remove the line to match the published
336.                # algorithm.
337.
338.                self.step1ab()
339.                self.step1c()
340.                self.step2()
341.                self.step3()
342.                self.step4()
343.                self.step5()
344.                return self.b[self.k0:self.k+1]
345.
346.
347.    if __name__ == '__main__':
348.        p = PorterStemmer()
349.        if len(sys.argv) > 1:
350.            for f in sys.argv[1:]:
351.                infile = open(f, 'r')
352.                while 1:
353.                    output = ''
354.                    word = ''
355.                    line = infile.readline()
356.                    if line == '':
357.                        break
358.                    for c in line:
359.                        if c.isalpha():
360.                            word += c.lower()
361.                        else:
362.                            if word:
363.                                output += p.stem(word, 0,len(word)-1)
364.                                word = ''
365.                            output += c.lower()
366.                    print output,
367.                infile.close()
```

In this thesis, data pre-processing was developed on python-based program. The source code consists of 1 file that is false_fail_data_preprocessing.py. The source code as shown below.

```
import os,re
import xlwt
import math
import traceback
class excelHandler:
    def __init__(self):
        self.ezxf = xlwt.easyxf
        self.heading_xf = self.ezxf('font: bold on; align: wrap on, vert centre, horiz center')
        kinds = ['text','text','text','text','text','text']
        kind_to_xf_map = {
            'date': self.ezxf(num_format_str='yyyy-mm-dd'),
            'int': self.ezxf(num_format_str='#,##0'),
            'money': self.ezxf('font: italic on; pattern: pattern solid, fore-colour grey25',
                num_format_str='$#,##0.00'),
            'price': self.ezxf(num_format_str='#0.000000'),
            'text': self.ezxf(),
            }
        data_xfs = [kind_to_xf_map[k] for k in kinds]
    def write_xls(self,file_name, sheet_name, headings, data = []):
        book = xlwt.Workbook()
        sheet = book.add_sheet(sheet_name)
        rowx = 0
        for colx, value in enumerate(headings):
            sheet.write(rowx, colx, value)
        for row in data:
            rowx += 1
            for colx, value in enumerate(row):
                sheet.write(rowx, colx, value)
        book.save(file_name)
class logUtil():
    def __init__(self,numHD,numZN):
       self.numHD = numHD
       self.numZN = numZN
    def readTable(self,this_file,queryTable = {},state_name = "",className = 'N'):
        print 'Reading Table in file >>> %s' %this_file
        dirCnt =  0
        correct_data = []
        fileName = os.path.basename(this_file)
        filtered_data = {}
        file = open(this_file,'r')
        log_data = file.read()
        file.close()
        matchTableInfo    = []
        head_retry_data   = []
        head_ec_data      = {}
        last_head         = -1
        matchAllCTemp = re.finditer("Cert_Temperature_Deg_C\s*(?P<d_temp>\d+)",log_data)
        c_temp_list = []
        for matchItem in matchAllCTemp:
           c_temp_list.append( matchItem.group("d_temp"))
```

```
51.       cert_temp = c_temp_list[-1]
52.       matchAllDriveTemp = re.finditer("Drive_Temperature_Deg_C\s*(?P<d_temp>\d+)
    ",log_data)
53.       d_temp_list = []
54.       for matchItem in matchAllDriveTemp:
55.          d_temp_list.append(int(matchItem.group("d_temp")))
56.
57.       last_d_temp  = d_temp_list[-1]
58.       if len(d_temp_list):
59.          mean_d_temp  = sum(d_temp_list) / len(d_temp_list)
60.       else:
61.          mean_d_temp = -1
62.       if state_name != "":
63.          match_state = re.search("^.*?AFH1 : BEGIN.*?\n(?P<data>.*?)(\*\*\*\*\*\*
    FAULT DETECTED \*\*\*\*\*|: COMPLETE.*?\n){1}",log_data, re.M | re.DOTALL)
64.          if not match_state:
65.             print "Can not found AFH1 state data"
66.             return None
67.          first_state_match_data = match_state.group('data')
68.          for thisTable in queryTable.keys():
69.                for match in re.finditer( "^%s:\s*(?P<header>.*\n)(?P<data>.*\n)*
    ?\n" %thisTable , first_state_match_data , re.M):
70.                   matchTableInfo.append(log_data.count("\n",0,match.start())+1)

71.                   matchTableInfo.append(match.start())
72.                   matchTableInfo.append(match.end())
73.                   header = match.group('header')
74.                   this_table_data = first_state_match_data[match.start():match.e
    nd()].split(match.group('header'))[1].rstrip()
75.                   test_stat_patt = ".*?F I N I S H E D\s+Testing st\((?P<test_nu
    m>\d+)\),\s+Test Stat:\s+(?P<test_stat>\d+),\s+Test Time:\s+(?P<test_time>[\d\.]
    +)"
76.                   log_after_thisTable = first_state_match_data[match.end():]
77.                   header_list = re.findall("\s*(?P<header_value>\S+)",header,re.
    S)
78.                   primary_key   = ['HD_PHYS_PSN','DATA_ZONE']
79.                   for round,line in enumerate(this_table_data.split("\n")):
80.                      for selectColumn in queryTable[thisTable].keys():
81.                         match_data = re.findall("\s*(?P<data_value>\S+)",line,re
    .S)
82.                         current_head          = match_data[header_list.index('HD
    _PHYS_PSN')]
83.                         current_zone          = match_data[header_list.index('DA
    TA_ZONE')]
84.                         current_active_heater = match_data[header_list.index('AC
    TIVE_HEATER')]
85.                         current_data          = match_data[header_list.index(sel
    ectColumn)]
86.                         if round == 0 :
87.                            head_retry_data.append(int(current_head))
88.                            if int(last_head) == int(current_head): #RESET LAST D
    ATA
89.                               if not filtered_data.has_key(current_head):
90.                                  filtered_data[current_head] = {}
91.                               filtered_data[current_head][current_active_heater]
     = []
92.                            else:
93.                               last_head = current_head
94.
95.                            nearest_testStat_match = re.search(test_stat_patt,log
    _after_thisTable)
```

```
96.                                 if not head_ec_data.has_key(current_head):
97.                                    head_ec_data[current_head] = {'R':[],'W':[]}
98.                                 if nearest_testStat_match:
99.                                    if current_active_heater == "R":
100.                                             ec = nearest_testStat_match.group('test
    _stat')
101.                                             head_ec_data[current_head]['R'].append(
    ec)
102.                                          elif current_active_heater == "W":
103.                                             ec = nearest_testStat_match.group('test
    _stat')
104.                                             head_ec_data[current_head]['W'].append(
    ec)
105.                                    else:
106.                                       if current_active_heater == "R":
107.                                          head_ec_data[current_head]['R'].append(
    -1)
108.                                       elif current_active_heater == "W":
109.                                          head_ec_data[current_head]['W'].append(
    -1)
110.                                 for filterCond in queryTable[thisTable][selectCo
    lumn]:
111.                                    filter_coloumn,filter_value = filterCond.spli
    t("=")
112.                                    if match_data[header_list.index(filter_coloum
    n)] == filter_value:
113.                                       if not filtered_data.has_key('%s' %curr
    ent_head):
114.                                          filtered_data[current_head] = {}
115.                                       if not filtered_data[current_head].has_
    key(current_active_heater):
116.                                          filtered_data[current_head][current_
    active_heater] = []
117.                                       filtered_data[current_head][current_act
    ive_heater].append(current_data)
118.              allHDReties = 0
119.              retry_by_head = {}
120.              failHDList = {}
121.              retryLimit = 4
122.              for the_hd in head_ec_data:
123.                 for the_activeHeater in head_ec_data[the_hd]:
124.                    if not retry_by_head.has_key(the_hd):
125.                       retry_by_head[the_hd] = {'R':0,'W':0}
126.                    ecList = head_ec_data[the_hd][the_activeHeater]
127.                    thisHDRetry = 0
128.                    for ec in ecList:
129.                       if int(ec) == 0:
130.                          break
131.                       elif int(ec) > 0:
132.                          thisHDRetry = thisHDRetry + 1
133.                          retry_by_head[the_hd][the_activeHeater] = retry_by_hea
    d[the_hd][the_activeHeater] + 1
134.                          allHDReties = allHDReties + 1
135.                    else:
136.                       if thisHDRetry >= retryLimit:
137.                          if not failHDList.has_key(the_activeHeater):
138.                             failHDList[the_activeHeater] = []
139.
140.                          failHDList[the_activeHeater].append(the_hd)
141.           if "_" in fileName:
```

```
142.                pattFileName = "(?P<SN>.*?)_(?P<OPER>.*?)_(?P<TS>.*?)_(?P<EC>.*
    ?).txt"
143.             else:
144.                pattFileName = "(?P<SN>.*?)\.(?P<TS>.*?)\.(?P<OPER>.*?)\.(?P<EC
    T>.*?).txt"
145.             match = re.search(pattFileName,fileName)
146.             try:
147.                SN_TS = [match.group("SN"),int(match.group("TS"))]
148.             except:
149.                SN_TS = [match.group("SN"),match.group("TS")]
150.             this_SN_row = []
151.             for the_activeHeater in failHDList:
152.                for failHD in failHDList[the_activeHeater]:
153.                   try:
154.                      this_SN_row.append( SN_TS + [className] + [str(failHD)] +
     [str(cert_temp)] + [str(mean_d_temp)] + [the_activeHeater] + [str(len(failHDLis
    t[the_activeHeater]))] + [str(allHDReties)] + head_ec_data[failHD][the_activeHea
    ter][-4:] + filtered_data[failHD][the_activeHeater ])
155.                   except:
156.                      print traceback.format_exc()
157.             return this_SN_row
158.       if __name__ == '__main__':
159.          numHD = 10
160.          numZN = 24
161.          cUtil = logUtil(numHD,numZN)
162.          fail_logfile_path = 'D:\\Learning\\MasterDegree\\Thesis\\Thesis_Exper
    iment_Data\\MEG_14703\\_TRUE_FAIL'
163.          pass_logfile_path = 'D:\\Learning\\MasterDegree\\Thesis\\Thesis_Exper
    iment_Data\\MEG_14703\\_FAKE_FAIL'
164.          initHeader     = ["SN","TS","STATUS","HD_FAIL","CERT_TEMP","MEAN_DRIV
    E_TEMP","FAIL_MODE","NUM_HD_FAIL_SAME_MODE","ALL_135_RETRY","RETRY#1","RETRY#2",
    "RETRY#3","RETRY#4"]
165.          headerByHDZN   = []
166.          attrByHD = []
167.          attrByZN       = ["FAIL_HD_RD_CNTCT_DAC"]
168.          attrByHDZN     = []
169.          for paramName in attrByHD + attrByHDZN:
170.             for numHead in range(numHD):
171.                if paramName in attrByHD:
172.                   headerByHDZN.append("HD_%s-%s" %(numHead,paramName))
173.                elif paramName in attrByHDZN:
174.                   for numZone in range(numZN):
175.                      headerByHDZN.append("HD_%s-ZN_%s-
    %s" %(numHead,numZone,paramName))
176.          for paramName in attrByZN:
177.             for numZone in range(numZN):
178.                   headerByHDZN.append("ZN_%s-%s" %(numZone,paramName))
179.          excell_header = initHeader + headerByHDZN
180.          data_fail = []
181.          data_pass = []
182.          for aFile in os.listdir(fail_logfile_path):
183.              logfile = os.path.join(fail_logfile_path,aFile)
184.              if os.path.isfile(logfile):
185.                this_log = os.path.join(fail_logfile_path,logfile)
186.                row_data = cUtil.readTable(logfile,{'P135_FINAL_CONTACT':{'RD_C
    NTCT_DAC':['MSRD_INTRPLTD=I']}},'AFH1','F')
187.                data_fail.extend( row_data )
188.          for aFile in os.listdir(pass_logfile_path):
189.              logfile = os.path.join(pass_logfile_path,aFile)
190.              if os.path.isfile(logfile):
191.                this_log = os.path.join(pass_logfile_path,logfile)
```

```
192.                row_data = cUtil.readTable(logfile,{'P135_FINAL_CONTACT':{'RD_C
    NTCT_DAC':['MSRD_INTRPLTD=I']}},'AFH1','P')
193.                data_pass.extend( row_data )
194.          data = data_pass + data_fail
195.          excell_filename = os.path.join(os.path.join(os.path.dirname(__file__)
    ,'Result'),'Thesis_14703_data.xls')
196.          excelHandler().write_xls(excell_filename ,"Raw Data" , excell_header
    , data)
197.          print "finished preparing data"
```

AUTHOR BIOGRAPHY

Name-Surname:	Mr. Natthakritta Rungtalay
Date of Birth:	June 14th, 1987
Present Address:	860/10, Moo 11, Tambol Sungnoen, Amphur Sungnoen, Nakornratchasima, Thailand 30170
Education:	2005-2009: Bachelor degree in Computer Engineering, Suraneree University of Technology.
Scholarships:	2005-2009: Scholarship for study in Bachelor degree in Computer Engineering by Suraneree University of Technology. 2011-2012: Scholarship for study in Master of Engineering in Data Storage Technology (English program) by NSTDA, KMITL and Seagate Technology (Thailand) Ltd.
Publications:	Natthakritta R. Chanon W., "AN ALTERNATIVE SOLUTION FOR HARD DISK DRIVE CLASSIFICATION PROCESS IMPROVEMENT", The 29th International Technical Conference on Circuit/Systems Computers and Communications (ITC-CSCC), July 1-4, 2014.

Experience:

2008-2009	Nextwaver.net Co.,Ltd. - Programmer
2009-Present	Seagate Technology (Thailand) Ltd. - Asia Engineering Firmware/Test