Determining the Accuracy of Budgets
A Machine Learning Application
for Budget Change Pattern Recognition

Kai Leung Yip

A thesis submitted in partial fulfillment

of the requirements for the degree of

Master of Computing

Unitec Insitute of Technology, 2012

9th November 2012

The Decentralized Machine Learning Intelligence (DMLI)
Laboratory

Primary Supervisor: Dr. Paul S. Pang

Secondary Supervisor: Dr. Xiaohui Zhao

Abstract

In New Zealand, growing a small and medium-sized enterprise (SME) has long been challenging. It is even more difficult without good business planning. Budget forecasting is a cornerstone of a business plan. The more accurate the budget is, the more likely the business plan can be achieved.

Creating budgets can be daunting and time consuming for SME owners. With the aid of modern budgeting software, non-financial professionals can create budgets without the hassle of managing error-prone spreadsheets. However, these budgeting softwares often provide no analysis or guidelines on the accuracy of the budgets. To mitigate this, this research develops an SME intelligent budgeting model that is able to help businesses and non-profit organisations make better decisions by improving forward looking financial analysis.

Research has shown that financial variables, such as cash flow, z-score and components of DuPont Analysis, can be used for the prediction of future events in SMEs. In this research, we investigate a SME in history the variation between the forecast (i.e., the budgets planned for future) and actual data (i.e., the corresponding realised values) on financial variables. For a specific SME industry (e.g., wine manufacturing industry), we are able to summarise and extract the characterised change pattern for that industry, and use the knowledge to label every historical forecast as a viable or inviable budget. Consequently, for SME intelligent budgeting, a classification model is developed in order to alert SME owners to budget inaccuracy and related risks. This allows them to immediately develop contingency measures, revise the policy and get the business back on the right track.

As an example of an SME industry group, we use data from 23 departments of New Zealandgovernment to build the above SME intelligent budgeting model. The software prototype is tested by a leave-one-out cross-validation approach. The result shows that the proposed model gives an overall 70.5% classification accuracy. Although the prediction is not perfect and is highly influenced by the training data, it demonstrates the existence of the change pattern of financial variables for a certain

SME industry group. Importantly, this opens the door for further investigation for industry-specified combination of financial variables for other groups of SMEs.

The proposed SME intelligent budgeting model is prototyped as a layer on top of Forecast 5, the 2012 version of budgeting software developed by Passage Software Limited Auckland, New Zealand. The prototype consists of a structured query language (SQL) database for storing Forecast 5 financial variables data, and an application programming interface (API) for its integration to Forecast 5. Through system integration, Forecast 5 is equipped to trigger an alarm whenever a risk of inaccuracy occurs in the budget setting of an SME. This has been a long-desired function needed by a business or a bank.

It is worth noting that sufficient forecast data collection is essential, when we are discovering the financial variable change patterns for a certain SME industry group. However, due to the ethical and confidential issues of company, the data collection in practice is extremely difficult and is often a long process. Thus, insufficient data is a major limitation of the presented research, and must be rectified before it can be used widely in different industries. Additionally, when new data is presented, the proposed model needs to be reconstructed from scratch. Thus, how to train the prediction model backwards and forwards incrementally remains a challenging problem and is left for future development.

Keywords — Budget Change Pattern Recognition, Budgeting, Business Planning, Data Mining, Financial Forecast, Financial Ratio, Financial Variable, Government Budget, Intelligent Budgeting, Machine learning, Small and Medium-sized Enterprise (SME), Small Business.

Acknowledgment

This research is based on a joint project between Unitec Institute of Technology and Passage Software aimed at improving innovative business software solution, funded by TechNZ Foundation of the Ministry of Science and Innovation (MSI), New Zealand. This research could not be completed without the support from Passage Software in the areas of data collection as well as technical and industrial advices.

I must acknowledge all the support given to me by the staff from Unitec Institute of Technology and project manager Stephen Cox from Innovation Associates. Lastly, special thanks are given to my supervisors Dr. Paul S. Pang and Dr. Xiaohui Zhao and all the contributive members of the DMLI research group of Department of Computing at Unitec Institute of Technology.

Passage Software, a specialist in accounting and budgeting software, are eager to find out the solution to the problem. By providing real industry data and knowledge, they collaborated with Unitec Institute of Technology on this research of SME intelligent budgeting.

K. L. Yip

Auckland

9th November 2012

Attestation of Authorship

"I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any other degree or diploma of a university or other institution of higher learning."

Contents

1 Introduction **1**

1.1 Research Background 1

1.2 Problem Definition 2

1.3 Literature Review 3

1.3.1 Cash Flow Analysis 3

1.3.2 z-score 4

1.3.3 DuPont Analysis 5

1.3.4 Data Mining 7

2 Methodology **9**

2.1 Definitions of Variables 9

2.2 System Design 12

2.2.1 Data Storage 12

2.2.2 Data Mining Software Prototype 14

2.3 Experiment Procedures 16

2.3.1 Data Transformation 16

2.3.2 Variable Calculation 17

2.3.3 Budget Accuracy Classification 18

2.3.4 Machine Learning and Cross Validation 19
2.3.5 Classification 20

3 Experiments and Results 22

3.1 Dataset Introduction 22
3.1.1 Data Collection 22
3.1.2 Data Preprocessing 24
3.2 Experimental Setup 25
3.3 Performance Evaluation 27
3.4 Experimental Results 29
3.4.1 Budget 2011/2012 30

4 Conclusions 33

4.1 Solution to Problem-solving 33
4.2 Contribution of the Research 34
4.3 Limitation and Future Development 34

Glossary 35

References 40

Appendices 41

A Source Codes 42

B Experiment Results 76

List of Figures

2.1 Relationship between Actual and Forecast Budgets 10

2.2 The Data Mining Software Prototype 14

3.1 Data Preprocessing (to be contined in Figure 3.2) 25

3.2 Machine Learning (to be contined in Figure 3.3) 26

3.3 Prediction 30

List of Tables

1.1 Comparison of financial analysis software in the U.S. 3

1.2 Four-stage DuPont Analysis 6

2.1 Budget Accuracy Criteria for Class Labelling 11

2.2 Definition of Financial Variables 11

2.3 Common Components 12

2.4 SQL Table `budgets` 13

2.5 SQL Table `companies` 13

2.6 Java Classes 15

2.7 Input Parameters for `calculateDeltas` 16

2.8 Contingency table 21

3.1 New ZealandGovernment Budgets 23

3.2 Selected Departments of New ZealandGovernment 23

3.3 Prediction Algorithms 26

3.4 Contingency Tables 28

3.5 Average Precision 28

3.6 Processing Time (in second) 29

3.7 Prediction Accuracy for NaiveBayes 30

3.8 Prediction Accuracy for ADTree 30

3.9 Prediction for Budget 2011/2012 Using NaiveBayes 31

3.10 Prediction for Budget 2011/2012 Using ADTree 31

A.1 Java Libraries 47

B.1 Cross Validation Summary for Net Profit Prediction 76

B.2 Cross Validation Summary for Cash Flow Prediction 76

B.3 Cross Validation Summary for z-score Prediction 77

B.4 Cross Validation Summary for Profit Margin Prediction 77

B.5 Cross Validation Summary for DuPont ROA Prediction 77

B.6 Cross Validation Summary for Return on Equity Prediction 77

List of Abbreviations

A	accuracy
$\boldsymbol{B}_k(t)$	actual budget date ended t of company k
$\boldsymbol{B}'_k(t)$	forecast budget date ended t of company k
$\boldsymbol{\Delta B}'_k(t)$	change of forecast budget date ended t of company k
$\Delta X'$	change of forecast financial variable
E	error or false positive rate
EBIT	earning before interest and tax
EBITA	earning before interest, taxes and amortization
F	F-measure
FL	financial leverage
FN	false negative
FP	false positive
k	company identifier
n	number of financial variable
NP	negative predictive rate
P	precision or positive predictive rate
PM	profit margin
R	recall rate or sensitivity of positive prediction
ROA	return on asset
ROE	return on equity
t	end date of budget
TN	true negative
TP	true positive
X	actual financial variable
X'	forecast financial variable
Y	test variable for class labelling

Chapter 1

Introduction

This chapter introduces related work on budgeting and business planning. The business enviornment in New Zealandand the special characteristic of SMEs are described in Section 1.1. Section 1.2 defines the problem that New ZealandSMEs are facing in terms of budgeting and the importance of accurate budgets. Section 1.3 reviews previous research about financial analyses and computational methods that inspire us to solve this problem.

1.1 Research Background

New Zealand, a strong free-trade advocate, is recognised as the fourth freest economy in the world and as number one for business freedom by the 2011 Index of Economic Freedom (The Heritage Foundation and The Wall Street Journal, 2011). New Zealandoffers a very consistently regulated environment for establishing businesses. In 2009, there were 476,558 SMEs, most of which hired less than 20 staff and employed over 30% (587,520 people) of the total workforce. SMEs in New Zealandconstitute over 90% of enterprises and are scattered across every industry classified by the Australian and New Zealand Standard Industrial Classification (ANZSIC)[1].

[1]developed by Statistics New Zealandand the Australian Bureau of Statistics to compile and analyse industry statistics in the two countries (Statistics New Zealand, 2010). Website: `http://www.stats.govt.nz/browse_for_stats/industry_sectors/anzsic06-industry-classification.aspx`

However, growing business substantially has long been the real challenge, this is evident from the decreasing survival rate of SMEs, where more business deaths than births are recorded. Generally, small firms are less likely to survive compared to larger firms. Only 31% of zero-employee enterprises born in 2001 survived to 2009. Just below half of firms with 1-5 employee born in 2001 endured into 2009. Whereas, half of firms with 100-499 employees at birth sustained into 2009, and 100% of firms with more than 500 employees born in 2001 survived to 2009 (Ministry of Economic Development, 2010).

Planning ahead is vital for all businesses to survive. Statistics show that more and more SMEs who have enjoyed positive revenue and income in the past are the ones who chose to have a written business plan in place for planning their businesses (VantagePoint, 2011). Budget forecasting has long been used by managers to provide better financial planning, and it is a crucial part of a business planning process to estimate future financial status of the company including profitability, liquidity and sustainability. The more accurate the forecast budget is, the more likely the business plan can be achieved.

However, surprisingly, a comparison (Estes & Savich, 2011) among accounting software in the United States (Table 1.1) reveals that all the software under review offer very limited functions to help small business owners to evaluate how accurate or viable a forecast budget is. Whereas, in New Zealand, Indigo, MYOB and Passage are the leaders of business software, most of their products are capable of forecasting and budgeting, but none of them has featured budget evaluation functions that can determine the accuracy of a budget.

1.2 Problem Definition

Due to the size of SMEs, a simply risky decision or an unrealistic forecast budget may significantly alter the business performance and change the direction a company (Priester & Wang, 2009) in future. Therefore, an objective measurement that can immediately identify the accuracy or the achievability of a forecast budget is needed by managers.

Research has shown that financial variables, such as cash flow, z-score and compon-

Table 1.1: *Comparison of financial analysis software in the U.S.*

Name	Developer	Strengths	Weaknesses
ACCPAC CFO	Sage	KPI, Sales, cash flow analysis, what if	Lack of industry benchmark, ranking, break-even analysis, z-score
BizBench	MBA Ware	Industry benchmark, ranking, what if	Lack of break-even, cash flow analysis, z-score
Financial Analysis	Thomson Reuters	Industry benchmark, ranking	Lack of break-even, cash flow analysis, z-score, what if
iLumen	iLumen	Industry benchmark, ranking, cash flow analysis	Lack of break-even analysis, z-score, what if
PlanGuru	New Horizon	Industry benchmark, ranking, cash flow analysis, what if, z-score	Lack of break-even analysis
ProfitCents	Sageworks	Industry benchmark, cash flow analysis, what if, z-score, multi-lingual	Lack of industry ranking
ProSystem fx Profit Driver	CompleteTax	Industry benchmark, ranking, cash flow analysis, what if	Lack of break-even analysis, z-score

ents of DuPont Analysis, can be used for the prediction of future events in SMEs (Nissim, 2001; Beaver, McNichols & Rhie, 2005; Soliman, 2008; Sun & Li, 2009; Collier, McGowan & Muhammad, 2010; Lin, Liang & Chen, 2011). In this research, we investigate a SME in history the variation between the forecast (i.e., the budgets planned for future) and actual data (i.e., the corresponding realised values) on financial variables. For a specific SME industry (e.g., wine manufacturing industry), we are able to summarise and extract the characterised change pattern for that industry, and use the knowledge to label every historical forecast as a viable or inviable budget. Consequently, for SME intelligent budgeting, a classification model is developed in order to alert SME owners for budget inaccuracy and its risk. This allows them to immediately develop contingency measures, revise the policy and get the business back on the right track. Previous research is reviewed in order to summarise the financial variables and methods that can be used to classify the accuracy of budgets.

1.3 Literature Review

1.3.1 Cash Flow Analysis

Although, profit (or its synonyms: earning, income and revenue) is the simplest way to determine the sustainability of a business, many other factors are hidden. It is because profit may not be presented in the form of cash but remaining outstanding, due from the customers in the form of accounts receivables according to generally accepted accounting principles (GAAP). Cash does not equal profit. It normally refers to the liquid assets in the bank account less any current short-term debts.

A company can have a large amount of income in terms of accounts receivables which are reported on income statement and as equity on balance sheet. However, the company can still be short on cash to operate, including purchasing and paying wages. This is because accounts receivable or other current assets may not be converted into cash immediately.

Despite having positive income and a positive net worth, a company can be considered as technically bankrupt if it fails to settle any current liabilities. Cash is king is a widely used expression to describe the importance of sufficient cash in a business, especially when a lot of companies facing financial distresses or even bankruptcy during recession.

A cash flow forecast aims to evaluate a company's future financial liquidity using financial models such as pro-forma statements. Cash flow forecast is an important tool to not only work out budgets, but also to determine the capital structure and discover weaknesses. However, cash flow projections are not primarily used as a tool to construct elaborate mathematical models or predict the future, but rather to identify potential risks in order to maximise the lender's interests (Fight, 2006).

In spite of their potentially lucrative nature, SMEs are generally considered the higher-risk sector. The greatest problem facing SMEs in growth is insufficient liquidity (Fight, 2006). Therefore, liquidity and cash flow analyses are also crucial for business planning as well as for lenders to make decision.

1.3.2 z-score

Altman's z-score is a multivariate approach for bankrupt and financial failure prediction. The coefficients of the variable was proposed by Altman (1968) aiming at best distinguishing financial healthy firms from distressed firms:

$$z = 1.2x_1 + 1.4x_2 + 3.3x_3 + 0.6x_4 + 1.0x_5 \tag{1.1}$$

$$\begin{aligned} \text{where } x_1 &= \frac{\text{working capital}}{\text{total assets}} \\ x_2 &= \frac{\text{retained earnings}}{\text{total assets}} \\ x_3 &= \frac{\text{EBITA}}{\text{total assets}} \\ x_4 &= \frac{\text{market value of equity}}{\text{total liabilities}} \\ x_5 &= \frac{\text{sales}}{\text{total assets}} \end{aligned}$$

However, these coefficients are not suitable for all environments. Modified versions of the model are found in the literature. Taffler z-score (1.2) is one version customised for the United Kingdom (Pogue, 2008).

$$z = c_0 + c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4 \tag{1.2}$$

$$\begin{aligned} \text{where } x_1 &= \frac{\text{profit before tax}}{\text{current assets}} \\ x_2 &= \frac{\text{current assets}}{\text{current liabilities}} \\ x_3 &= \frac{\text{current liabilities}}{\text{total assets}} \\ x_4 &= \text{no credit interval} \end{aligned}$$

The z-score models are empirically tested to be adaptable for the real situations. In addition, z-score combined with contemporary artificial intelligence (AI) method gives better bankrupt and financial failure prediction.

1.3.3 DuPont Analysis

DuPont Analysis is developed by F. Donaldson Brown, a staff of DuPont Corporation in 1920's. Since then, it has been widely used to describe the strengths and weaknesses of a company in the areas of profitability, operating efficiency and financial efficiency using financial ratios: profit margin (PM), return on asset (ROA) and financial leverage (FL) respectively. The DuPont formula rewrites return on equity (ROE) as follows:

$$\begin{aligned} \text{ROE} &= \text{PM} \times \text{ROA} \times \text{FL} \\ &= \frac{\text{Net Profit}}{\text{Sales}} \times \frac{\text{Sales}}{\text{Assets}} \times \frac{\text{Assets}}{\text{Equity}} \end{aligned} \tag{1.3}$$

The DuPont's components in three fundamental aspects: sales volume, profit margin and financial leverage, are found empirically to be interacting with each other. Priester and Wang (2009) developed a four-stage DuPont analysis in order to pinpoint the factors and the reasons for a company's performance as illustrated in Table 1.2.

Table 1.2: *Four-stage DuPont Analysis*

Stage I	Stage II	Stage III	Stage IV	
	ROA = $\frac{\text{Net Sales}}{\text{Total Assets}}$	PM = $\frac{\text{Net Profit}}{\text{Net Sales}}$	FL = $\frac{\text{Total Assets}}{\text{Total Equity}}$	= ROE
Year 1 Year 2 ⋮				
	Look at: Receivables Turnover Inventory Turnover Fixed Asset Turnover To determine: Asset Productivity	Look at: Gross Profit Margin Operating Cost Margin Operating Profit Margin To determine: If there is a Pricing Margin Squeeze	Look at: Interest Coverage and Interest Cost and/or Fixed Charge Coverage which includes Lease To determine: Payments + Principle Repayments, also Short-term Debt vs. Long-term Debt Mix	

This is how Priester and Wang (2009) four-stage DuPont analysis works:

1. Gain a broad understanding of these factors and their relationship to each other, looking for the interaction of change between one factor to another. For example, is sales volume improved bought at the expense of the weakening profit margin factor?
2. Look at the overall efficiency by investigating the average number of days of receivables to be collected, inventories to be sold and merchandise to be paid respectively.
3. Look at the factors that affect the DuPonts net profit margin, including gross profit margin, operational efficiency ratio, operating profit margin and interest burden.
4. Look at the financial leverage multiplier, determining whether the change of the financial leverage is manipulated deliberately, i.e. raising debt for investment or expansion or giving out generous dividend or just caused by chance, i.e. interest rate change. As, this may eventually cause trouble for the company.

Despite the age of DuPont Analysis, it is widely used for financial analysis and forecast. By using DuPont Analysis, Soliman (2008) found that current asset turnover was positively correlated to future return on net operating assets. Collier et al. (2010) also showed how Asian financial crisis impacted on the financial performance of a company. This shows that DuPont analysis can help SME to better estimate the future performance and risk of their business. However, DuPont analysis is not transferrable to another industry, It is only meaningful when the comparison is made against a benchmark, a peer competitor or a company's own history. This inspires us to develop an industry-specified budget accuracy classification model.

1.3.4 Data Mining

Due to the complex interrelationships of the economic and financial dimensions of a company, Gallizo, Gargallo and Salvador (2008) showed, in contrast with the conventional univariate analysis, the need for a multivariate approach. However, looking for an explicit formula to determine the accuracy of budget is still almost impossible. Data mining is a good approximation approach to the solution of this kind of problem.

Odom and Sharda (1990) (Shin, Lee & Kim, 2005) were the first to attempt to use a neural network (NN) for bankruptcy prediction. Later on, NNs were overwhelmed by support vector machine (SVM) for these kinds of problems, as SVM is superior in terms of speed, versatility, and regards the possibility of incremental learning (Ozawa, Pang & Kasabov, 2008). Haardle, Moro, & Schaafer (2003) (Shin et al., 2005) and J. Min and Lee (2005) found that SVM outperformed other models such as NN, multiple discriminate analyse (MDA) and learning vector quantisation (LVQ). By combining the use of linear discriminate functions, SVM was found to have the ability to learn from relatively small amount of data. S. Min, Lee and Han (2006) later improved a SVM-based model by the means of a integrating generic algorithm (GA) and SVM. Also, Hua, Wang, Xu, Zhang and Liang (2007) demonstrated how their proposed integrated binary discriminant rule (IBDR) method performed better than the conventional SVM for prediction problems in small businesses.

Chapter 2

Methodology

Given historical data of a company, including actual and forecast budgets, the accuracy of the forecast budgets can be evaluated using the proposed method explained in this chapter. Section 2.1 introduces the definitions of budgets and financial variables. Section 2.2 explains the overall design of the system. Section 2.3 describes the detail procedures including data collection, data preprocessing, data, variable calculation, machine learning and prediction.

2.1 Definitions of Variables

Let k be the index of a company and t be the time period covered by a budget. We define that $\boldsymbol{B}_k(t)$ and $\boldsymbol{B'}_k(t)$ represent the actual and forecast budget respectively,

$$\boldsymbol{B}_k(t) = [X_1(t), X_2(t), \ldots, X_n(t)], \tag{2.1}$$

$$\boldsymbol{B'}_k(t) = \left[X_1^{'}(t), X_2^{'}(t), \ldots, X_n^{'}(t)\right], \tag{2.2}$$

where $X_n(t)$ and $X_n^{'}(t)$ are the actual and forecast financial variable of company k respectively. n is the total number of financial variables as the list given in Table 2.2 at the end of this section.

Let $\Delta X_n^{'}(t)$ be the proportional change of a forecast financial variable $X_n^{'}(t)$ to its

previous actual $X_n(t-1)$,

$$\Delta X_n^{'}(t) \;=\; \frac{X_n^{'}(t)}{X_n(t-1)} - 1, \tag{2.3}$$

and let $\boldsymbol{\Delta B}^{'}{}_k(t)$ be the change of a forecast budget,

$$\boldsymbol{\Delta B}^{'}{}_k(t) \;=\; \left[\Delta X_1^{'}(t), \Delta X_2^{'}(t), \ldots, \Delta X_n^{'}(t)\right]. \tag{2.4}$$

The vector $\boldsymbol{\Delta B}^{'}{}_k(t)$ is given a class label $\boldsymbol{A}_k(t)$ by the following rules,

$$\boldsymbol{A}_k(t) = \left[\begin{array}{l} 0 : Y(t) < Y^{'}(t) \\ 1 : Y(t) \geq Y^{'}(t) \end{array}\right], \tag{2.5}$$

where $Y(t)$ is the actual value of a test variable. By comparing with its forecast value $Y^{'}(t)$, a label '0' is given if the actual budget under-performs the forecast (actual $<$ forecast). Whereas, a label '1' is given if the actual budget outperforms the forecast (actual $\geq$ forecast).

Figure 2.1 illustrates the interrelationship between $\boldsymbol{B}_k(t)$, $\boldsymbol{B}^{'}{}_k(t)$ and $\boldsymbol{A}_k(t)$ with respect to t.

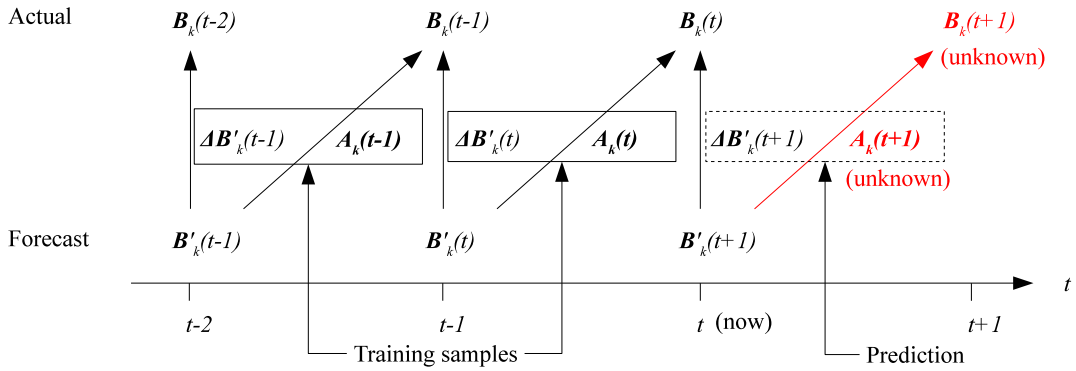

Figure 2.1: *Relationship between Actual and Forecast Budgets*

According to the literature review in Section 1.3, we summarise the financial variables which determine the future performance and sustainability of SME businesses. We choose 6 different financial variables as the criteria for labelling $\boldsymbol{\Delta B}^{'}{}_k(t)$: (C1) net profit, (C2) cash flow, (C3) z-score, (C4) profit margin, (C5) DuPont ROA and (C6)

ROE as shown in Table 2.1. Using the rules in 2.5, the class label $\boldsymbol{A}_k(t)$ returns either '0' or '1'.

Table 2.1: *Budget Accuracy Criteria for Class Labelling*

Criterion	Test Variable
C1	Net Profit = sales - cost of goods - operating cost - interest - tax
C2	Cash Flow
C3	z-score $= 1.2x_1 + 1.4x_2 + 3.3x_3 + 0.6x_4 + 1.0x_5$ (see Equation (1.1))
C4	Profit Margin = net profit / sales
C5	DuPont ROA = sales / total assets
C6	Return on Equity = net profit / equity

Take C1 in Table 2.1 as an example, if the actual net profit is greater or equals to (outperforms) its forecast value, a class label '1' is given, otherwise (under-performs) a class label '0' is given.

2.2 System Design

A data mining software prototype is developed using free resources such as the GNU General Public License and open-source software. The industry-standard cross-platform SQL database (MySQL Server 5.5.3) and programming language (Java 1.6.0_31) are employed, so that the software prototype can be easily implemented regardless of computer architecture. For software integration, an API is designed as an interface to existing financial software.

The software prototype is developed on a Mac OSX 10.7.4 machine (Intel Core i5 2.7GHz, 12GB RAM). Source codes are listed in Appendix A.

2.2.1 Data Storage

Although, the financial variables X_1 to X_{19} defined in Table 2.2 are not presented directly from the financial statements of a budget. They can be derived from common components available in the statements. These common components are summarised in Table 2.3.

Table 2.2: *Definition of Financial Variables*

Variable	Definition	Mentioned by
X_1	cash ratio = $\frac{\text{cash+cash equivalent}}{\text{current liabilities}}$	Beaver (1966), Zmijewski (1984), Martens, Bruynseels, Baesens, Willekens and Vanthienen (2008)
X_2	cash flow/total debt	Beaver (1966), Deakin (1972), Blum (1974), Zmijewski (1984), Martens et al. (2008)
X_3	cash flow/total asset	Deakin (1972), Ohlson (1980)
X_4	cash flow/sales	Deakin (1972), Li and Sun (2009)
X_5	debt ratio = $\frac{\text{total liabilities}}{\text{total assets}}$	Beaver (1966), Deakin (1972), Blum (1974), Zmijewski (1984), Martens et al. (2008)
X_6	working capital[a]/total asset	Beaver (1966), Altman (1968)
X_7	market value equity/total debt	Altman (1968), Martens et al. (2008), Li and Sun (2009)
X_8	current assets/total asset	Deakin (1972)
X_9	quick asset[b]/total asset	Deakin (1972)
X_{10}	asset turnover = $\frac{\text{net sales}^c}{\text{total assets}}$	Altman (1968), Martens et al. (2008), Li and Sun (2009)
X_{11}	current debt/sales	Deakin (1972)
X_{12}	quick asset/sales	Deakin (1972)
X_{13}	working capital/sales	Beaver (1966), Deakin (1972), Ohlson (1980), Martens et al. (2008)
X_{14}	net income/total asset	Beaver (1966), Deakin (1972), Ohlson (1980), Zmijewski (1984)
X_{15}	retained earnings/total asset	Altman (1968), Ding, Song and Zen (2008)
X_{16}	EBIT[d]/total asset	Altman (1968), Li and Sun (2009)
X_{17}	PM = $\frac{\text{net profit}}{\text{sales}}$	Soliman and Lundholm (2003), Soliman (2008), Priester and Wang (2009), Collier et al. (2010), Narayanan (2010)
X_{18}	ROA = $\frac{\text{sales}}{\text{assets}}$	Altman (1968), Soliman and Lundholm (2003), Soliman (2008), Priester and Wang (2009), Collier et al. (2010), Narayanan (2010)
X_{19}	FL = $\frac{\text{assets}}{\text{equity}}$	Soliman and Lundholm (2003), Soliman (2008), Priester and Wang (2009), Collier et al. (2010), Narayanan (2010)

[a]working capital = current assets - current liabilities
[b]quick assest = current assets - inventories
[c]net sales = sales - cost of goods
[d]earning before interest and tax (EBIT) = sales - cost of goods - operating costs

Table 2.3: *Common Components*

Variable	Available in	Required by
cash equivalent	balance sheet	X_1
current liabilities	balance sheet	X_1, X_5, X_6, X_{11}
non-current liabilities	balance sheet	X_1, X_5
current debts	balance sheet	X_2, X_7
non-current debts	balance sheet	X_2, X_7
current assets	balance sheet	X_3, X_5,X_6, X_8, X_9, X_{10}, X_{14}, X_{15}, X_{16}, X_{18}, X_{19}
non-current assets	balance sheet	X_3, X_5,X_6, X_8, X_9, X_{10}, X_{14}, X_{15}, X_{16}, X_{18}, X_{19}
inventory	balance sheet	X_9
market value equity	balance sheet	X_7
total equity	balance sheet	X_{19}
retained earnings	balance sheet	X_{15}
cash flow	cash statement	X_2, X_3, X_4
total sales	income statement	X_{10}, X_{16}, X_{17}, X_{18}
cost of goods	income statement	X_{10}, X_{16}
operating costs	income statement	X_{16}
interest costs	income statement	X_{17}
tax	income statement	X_{17}

In order to store these common components, a SQL table `budgets` is created according to the specification listed in Table 2.4. Values are stored in one thousand New Zealanddollar units in this table.

The field `company_id` of table `budgets` referes to another table `companies` which stores the company information as the specification listed in Table 2.5. The above data tables and schema are created by an SQL script `createTables.sql` shown in Listing A.1 of Appendix A.

2.2.2 Data Mining Software Prototype

In order to process the data as discussed in Section 2.2.1, a data mining software prototype is developed which consists of three main parts: (1) the SQL database tables, (2) an SQL stored procedure `calculateDeltas` and, (3) a Java programme. The overall structure of the software prototype is illustrated in Figure 2.2.

In order to transform raw data to the SQL database, A Java interface `DataCollector` is designed for handling different data source. `DataCollector` consists of methods which are called by Java class `BudgetSubmitter` to extract common components (mentioned in Section 2.2.1) in the following steps:

Table 2.4: *SQL Table* `budgets`

Column	Storage Type	Description
`budget_id`	`INT`	Budget unique ID
`company_id`	`INT`	Company unique ID for joining with company table
`periodicity`	`DECIMAL(2,1)`	Periodicity of the budget in unit month (1=monthly, 3=quarterly, 12=annually)
`end_date`	`DATE`	The end date of the budget period
`actual`	`TINYINT(1)`	Either 0=forecast or 1=actual
`cash_eq`	`DOUBLE`	cash equivalent
`cur_liab`	`DOUBLE`	current liabilities
`non_cur_liab`	`DOUBLE`	non-current liabilities
`cur_debt`	`DOUBLE`	current debts
`non_cur_debt`	`DOUBLE`	non-current debts
`cur_asset`	`DOUBLE`	current assets
`non_cur_asset`	`DOUBLE`	non-current assets
`inventory`	`DOUBLE`	inventory
`market_equity`	`DOUBLE`	market value equity
`equity`	`DOUBLE`	total equity
`ret_earning`	`DOUBLE`	retained earnings
`cash_flow`	`DOUBLE`	cash flow
`sales`	`DOUBLE`	total sales
`cog`	`DOUBLE`	cost of goods
`oper_cost`	`DOUBLE`	operating costs
`interest`	`DOUBLE`	interest costs
`tax`	`DOUBLE`	tax

Table 2.5: *SQL Table* `companies`

Column	Storage Type	Description
`company_id`	`INT`	Company unique ID
`company_name`	`VARCHAR(100)`	Compnay name
`country`	`VARCHAR (50)`	Country or region
`size`	`INT`	Employee size
`anzsic`	`VARCHAR(20)`	Unique ID of the budget period

1. The Java interface `DataCollector` collects data from the data source.
2. Create Java instances of `ActualBudget` and `ForecastBudget` respectively.
3. `BudgetSubmitter` submit budget data to the database by calling methods in interface `DataCollector`.

Secondly, the SQL stored procedure `calculateDeltas` returns training sets from the data stored in the database in the following steps:

1. Query the common financial variables from the SQL table `budgets` by given the input parameters as described in Table 2.7.
2. Input attributes are evaluated by the definition in (2.3).

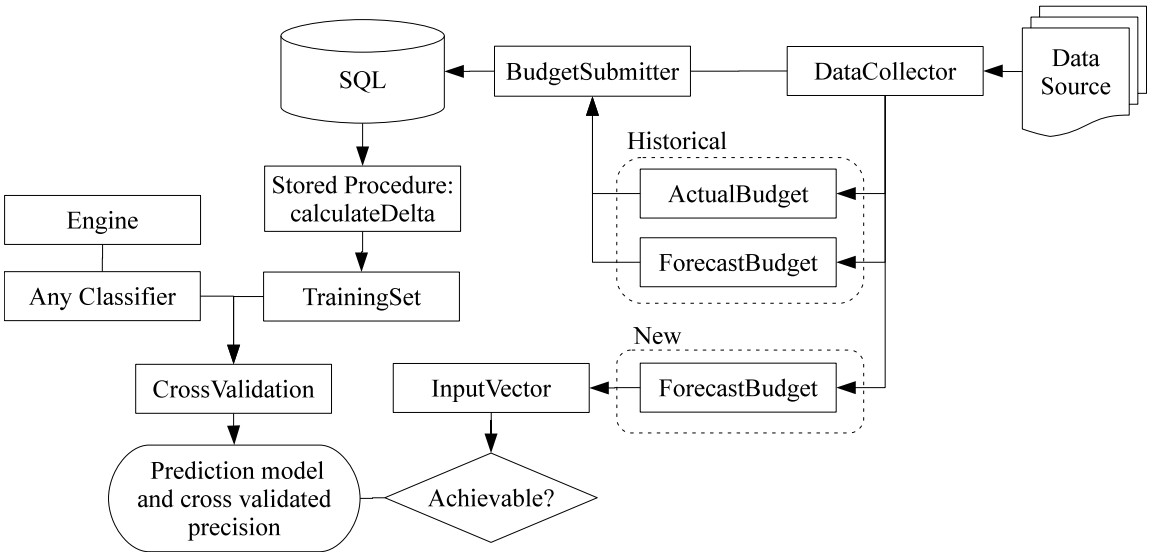

Figure 2.2: *The Data Mining Software Prototype*

Table 2.6: *Java Classes*

Package[a]	Class	Type	Description
`api`	`BudgetCalculator`	class	Methods to get budget data from the database.
	`BudgetSubmitter`	class	Methods to submit budget data to the database.
	`DataCollector`	interface	Methods required by BudgetSubmitter.
	`Main`	class	The main entry point.
`api.budget`	`Budget`	class	Budget identifier.
	`ActualBudget`	class	Sub-class of Budget with `actual=1`.
	`ForecastBudget`	class	Sub-class of Budget with `actual=0`.
	`Periodicity`	enum	`MONTHLY, QUARTERLY, ANNUALLY`.
`api.company`	`Company`	class	Company identifier.
	`Government`	class	Sub-class of Company with `anzsic=OO211`.
`api.engine`	`CrossValidation`	class	Methods for cross validation.
	`Engine`	abstract	Abstract class for all prediction engines.
	`InputVector`	class	Object that stores input attributes.
	`TrainingSet`	class	Collection of training vectors.
	`TrainingVector`	class	Sub-class of InputVector with a class label.
	`Weka`	abstract	Sub-class of Engine with Weka[b] 3.6.4 API.
`util`	`*Util`	class	Common static methods.
	`SqlClient`	class	SQL connection client.

[a] Java documentation can be found at `http://kuma.hopto.org/~antonyip/passage/api`

[b] an open-source data mining application developed by University of Waikato (Hall et al., 2009).

3. Output class label is classified by the definition in (2.5).
4. The training set is returned.

This SQL stored procedure is created by a SQL script `createProcedures.sql` as shown in Listing A.2 in Appendix A.

Table 2.7: *Input Parameters for* `calculateDeltas`

Parameter	Type	Usage[a]
`company`	`VARCHAR(20)`	Company index.
`anzsic`	`VARCHAR(20)`	ANZSIC code. E.g. `OO211` for government
`country`	`VARCHAR(3)`	Country abbreviation. E.g. `NZL` for New Zealand
`size_from`	`VARCHAR(6)`	Minimum employee size.
`size_to`	`VARCHAR(6)`	Maximum employee size.
`end_date_from`	`VARCHAR(10)`	Budgets since this date (yyyy-mm-dd).
`end_date_to`	`VARCHAR(10)`	Budgets until this date (yyyy-mm-dd).
`periodicity`	`DOUBLE`	Unit month. E.g. 1 (monthly), 3 (quarterly), 12(Annually).
`test`	`VARCHAR(500)`	A formula using the column names in table `budgets` to classify the budget accuracy. E.g. `cash_flow / sales`

[a]all parameters accept empty string to ignore the constraint except for periodicity

Thirdly, the returned training set is processed in the following steps:

1. Create an instance of `TrainingSet` which contains a set of `TrainingVector` from the data returned by the SQL stored procedures `calculateDeltas`.
2. Create an instance of `Engine` that contains the methods for machine learning.
3. Leave-one-out cross validation is performed by a field object `CrossValidation` of `Engine`.
4. The prediction model is trained by all samples and saved in binary format.
5. Prediction is made by calling the `predict()` method of class `Engine`.

2.3 Experiment Procedures

The data mining software prototype completes the experiment in the following steps:

1. Data Transformation
2. Variable Calculation
3. Budget Accuracy Classification
4. Machine Learning and Cross Validation
5. Classification

The following sections are the procedures further explained.

2.3.1 Data Transformation

According to Figure 2.2, given company budget data, the Java class `BudgetSubmitter` stores the data in the database by the following steps:

1. `BudgetSubmitter` is instantiated by the class constructor as shown in Listing 2.1. This constructor requires an implementation of the interface `DataCollector` as shown in Listing 2.2. A pesudo code example for the implementation of one of the abstract methods of `DataCollector` is shown in Listing 2.3.
2. Common financial variables are retrieved by the implemented methods of the interface `DataCollector`. `BudgetSubmitter` then submits these variables with company data to the database using some SQL statements as shown in Listing 2.4.

Listing 2.1: *Constructor for Java Class* `BudgetSubmitter`

```
1 public BudgetSubmitter(DataCollector dataCollector) {
2         this.dataCollector = dataCollector;
3 }
```

Listing 2.2: *Java Interface* `DataCollector`

```
1  public interface DataCollector {
2      boolean isAvailable(Budget budget);
3      double getCashEquivalents(Budget budget);
4      double getCurrentLiabilities(Budget budget);
5      double getNonCurrentLiabilities(Budget budget);
6      double getCurrentDebts(Budget budget);
7      double getNonCurrentDebts(Budget budget);
8      double getCurrentAssets(Budget budget);
9      double getNonCurrentAssets(Budget budget);
10     double getInventories(Budget budget);
11     double getMarketValueEquity(Budget budget);
12     double getEquity(Budget budget);
13     double getRetainedEarnings(Budget budget);
14     double getCashFlow(Budget budget);
15     double getSales(Budget budget);
16     double getCostOfGoods(Budget budget);
17     double getOperatingCost(Budget budget);
18     double getInterestCost(Budget budget);
19     double getTax(Budget budget);
20 }
```

Listing 2.3: *Pesudo Code Example for* `DataCollector` *Implementation*

```
1 double getCashEquivalents(Budget budget){
2         for each cell in budget.csv {
3                 if (cell.column == budget.year && cell.row == cash equivalents){
4                         return cell.value;
5                 }
```

```
6         }
7         return null;
8 }
```

Listing 2.4: *SQL Statement Example for Budget Storage*

```
1 REPLACE INTO `companies` (`company_id`, `company_name`, `anzsic`, `country`, `size` ) VALUES (1, 'ABC', 'A1111',
      'NZL', 50 );
2 REPLACE INTO `budgets` (`company_id`, `periodicity`, `end_date`, `actual`, `cash_eq`, `cur_liab`, `non_cur_liab
     `, `cur_debt`, `non_cur_debt`, `cur_asset`, `non_cur_asset`, `inventory`, `market_equity`, `equity`, `
     ret_earning`, `cash_flow`, `sales`, `cog`, `oper_cost`, `interest`, `tax` ) VALUES (1, 12, '2010-06-30', 1,
      100, 200, ... );
```

2.3.2 Variable Calculation

The SQL stored procedure `calculateDeltas` constructs a training set with columns ΔX_1 to ΔX_{19} and a class label in the following steps:

1. Call `calculateDeltas` with input parameters as described in Table 2.7. An example is shown in Listing 2.5.
2. According to the variable definitions in Table 2.2, X_1 to X_{19} are calculated from the common financial variables stored in the SQL table `budgets`. Actual and forecast values of the variables are stored in two temporary table `actual_ratios` and `forecast_ratios` respectively.
3. Partial code of `calculateDeltas` shown in Listing 2.6 matches the forecast ratios to their previous actual ratios.
4. ΔX_1 to ΔX_{19} are the differences between these forecast ratios and their previous actual ratios.

Listing 2.5: *SQL Statement Example for Calling* `calculateDeltas`

```
1 CALL `calculateDeltas` ('', '00211', 'NZL', 0, '', '2008-06-30', '2011-06-30', 12, 'sales - cog - oper_cost -
     interest - tax');
```

Listing 2.6: *Partial Code of* `calculateDeltas`

```
1 ...
2    (SELECT *, LAST_DAY(DATE_SUB(end_date, INTERVAL periodicity MONTH)) AS prev_end_date
3       FROM forecast_ratios
4     ) AS forecast_ratios
5     LEFT JOIN actual_ratios
6     ON (forecast_ratios.prev_end_date = actual_ratios.end_date
7     AND forecast_ratios.company_id = actual_ratios.company_id)
8 ...
```

2.3.3 Budget Accuracy Classification

Given a class label criterion, the SQL stored procedure `calculateDeltas` evaluates the class label of a budget in addition to the vector of variables ΔX_1 to ΔX_{19} in the following steps:

1. Partial code of `calculateDeltas` shown in Listing 2.7 matches forecast ratios to their corresponding actual ratios.
2. By the definitions in (2.5), a class label either 0 or 1 is given.
3. The procedure returns a 20-column training set with columns ΔX_1 to ΔX_{19} and a class label.

Listing 2.7: *Partial Code of* `calculateDeltas`

```
...
  (SELECT forecast.budget_id, (actual.criteria - forecast.criteria) AS diff
      FROM
      (SELECT budgets.*,criteria FROM `budgets` JOIN actual_ratios USING (budget_id)) AS actual
      RIGHT JOIN
      (SELECT budgets.*,criteria FROM `budgets` JOIN forecast_ratios USING (budget_id)) AS forecast
      USING (company_id, end_date)
  ) AS criteria_accuracy
...
```

2.3.4 Machine Learning and Cross Validation

The training set returned by the previous step is then processed by the Java part of the data mining software prototype by the following steps:

1. The training set is encoded to an instance of class `TrainingSet`, which contains a collection of `TrainingVector` objects as shown in Listing 2.8.
2. An instance of class `Engine` (Listing A.17) as shown in Listing 2.9.
3. `CrossValidation` (Listing A.16) creates new instances of `Engine` which runs an independent thread to perform as shown in Listing 2.10. Each independent thread excludes one sample from the training set and uses it for prediction.
4. The cross validation outputs are stored in the fields of a `CrossValidation` object. The precision is evaluated by the methods shown in Listing 2.11.

Listing 2.8: *Partial Code for* `TrainingSet` *Instantiation*

```
...
List<Object[]> list = new LinkedList<Object[]>(Arrays.asList(table));
List<TrainingVector> trainingSet = new LinkedList<TrainingVector>()
for (Iterator<Object[]> it = list.iterator(); it.hasNext();) {
```

```
            Object[] row = it.next();
            Object[] input = Arrays.copyOf(row, row.length - 1);
            Object output = row[row.length - 1];
            if (output == null) {
                it.remove();
            } else {
                trainingSet.add(new TrainingVector(input, output));
            }
}
...
```

Listing 2.9: *Partial Code for* **`Engine`** *Instantiation*

```
...
new Weka() {
        @Override
        public Class<? extends Classifier> getClassifierType() {
                return weka.classifiers.functions.LibSVM.class;
        }
}
...
```

Listing 2.10: *Partial Code for Leave-one-out Cross Validation*

```
...
for (final TrainingVector testSample : trainingSet) {
            final Engine engine = GeneralUtil.newInstance(engineType);
            new Thread(new Runnable() {
                public void run() {
                    TrainingSet newTrainingSet = new TrainingSet();
                    newTrainingSet.setId(trainingSet.getId());
                    for (TrainingVector trainingSample : trainingSet) {
                        if (trainingSample != testSample) {
                            newTrainingSet.add(trainingSample);
                        }
                    }
                    engine.train(newTrainingSet);
                    testSample.setPredictedOutput(engine.predict(testSample));
                    double actual = (Double) testSample.getDesiredOutput();
                    double predicted = (Double) testSample.getPredictedOutput();
                    if (actual >= 0.5) {
                        if (predicted >= 0.5) {
                            addTruePositive();
                        } else {
                            addFalseNegative();
                        }
                    } else {
                        if (predicted >= 0.5) {
                            addFalsePositive();
                        } else {
                            addTrueNegative();
                        }
                    }
                }
            }).start();
}
...
```

Listing 2.11: *Methods for Cross Validation Precision*

```
public Integer[][] getContingencyTable() {
        return new Integer[][]{
        new Integer[]{truePositive, falsePositive},
        new Integer[]{falseNegative, trueNegative}};
}
public double getAccuracy() {
```

```
7         return (double) (truePositive + trueNegative) / getSampleCount();
8 }
9 public double getPrecision() {
10        return (double) truePositive / (truePositive + falsePositive);
11 }
12 public double getNegativePrecision() {
13        return (double) trueNegative / (falseNegative + trueNegative);
14 }
15 public double getRecall() {
16        return (double) truePositive / (truePositive + falseNegative);
17 }
18 public double getError() {
19        return (double) falsePositive / (falsePositive + trueNegative);
20 }
21 public double getFMeasure() {
22        double P = getPrecision();
23        double R = getRecall();
24        return P * R * 2 / (P + R);
25 }
```

2.3.5 Classification

Contingency tables are widely used to assess a classification model by cross validation. A contingency table consists of the numbers of correct and incorrect predictions: True Positive (TP), False Positive (FP), False Negative (FN) and True Negative (TN) as shown in Table 2.8.

Table 2.8: *Contingency table*

	True Values	
	Positive	Negative
Predicted Positive	True Positive (TP)	False Positive (FP)
Predicted Negative	False Negative (FN)	True Negative (TN)

The performance of the classification is evaluated firstly through Accuracy A, the computation of the probabilities of correct predictions:

$$A = \frac{TP + TN}{TP + TN + FP + FN}. \tag{2.6}$$

For positive prediction (class: 1), the performance is evaluated through Precision P, the positive predictive rate:

$$P = \frac{TP}{TP + FP} \tag{2.7}$$

The recall rate or sensitivity of positive prediction,

$$R = \frac{TP}{TP + FN} \tag{2.8}$$

The Error E is the false positive rate. The lower the error, the better the classifier is.

$$E = \frac{FP}{FP + TN} \tag{2.9}$$

For negative prediction (class: 0), the performance is evaluated through Negative Precision NP, the negative predictive rate:

$$NP = \frac{TN}{FN + TN} \tag{2.10}$$

F-measure is used to measure prediction performance if the classes are imbalance,

$$F = \frac{2PR}{P + R} \tag{2.11}$$

Chapter 3

Experiments and Results

In our experiment, we use data from 23 departments of the New Zealandgovernment to build an SME intelligent budgeting model. Section 3.1 introduces the dataset and data preprocessing. Section 3.2 explains the development of the software prototype. Section 3.3 gives the preformance evaluation of the prediction model. Section 3.4 shows that the proposed model gives an overall 70.5% classification accuracy.

3.1 Dataset Introduction

3.1.1 Data Collection

Departmental budgets are downloaded from the Treasury of New ZealandGovernment at the links shown in Table 3.1. These budgets contain income statements, balance sheets and cash flow statements in the fiscal years 2008 to 2012. Each fiscal year starts from 1st July of the preceding year to 30th June. For example, budget 2008 covers the period from 1st July, 2007 to 30th June 2008. 23 departments with the highest value of average assets are selected for the experiment as shown in Table 3.2.

Table 3.1: *New ZealandGovernment Budgets*

Fiscal Year	Actual	Forecast	Download Link
2012	2010/2011	2012/2013	http://www.treasury.govt.nz/budget/2012/data/b12-dept-ffs-data.xls
2011	2009/2010	2011/2012	http://www.treasury.govt.nz/budget/2011/data/b11-dept-ffs-data.xls
2010	2008/2009	2010/2011	http://www.treasury.govt.nz/budget/2010/data/b10-dept-ffs-data.xls
2009	2007/2008	2009/2010	http://www.treasury.govt.nz/budget/2009/data/b09-dept-ffs-data.xls
2008	2006/2007	2008/2009	http://www.treasury.govt.nz/budget/2008/data/b08-dept-ffs-data.xls

Table 3.2: *Selected Departments of New ZealandGovernment*

Department	Average Assets ('000)	Average Surplus ('000)
Ministry of Education	10484585	545.2
Department of Corrections	2171996.4	22316.2
Ministry of Justice	761081.2	8682.2
Department of Conservation	558302.4	4492.4
Ministry of Foreign Affairs and Trade	488775.4	6094.8
Department of Internal Affairs	150370.8	8608.6
National Library of New Zealand	121785.5	3773.5
Department of Labour	115816.8	-224.8
Ministry for Primary Industries	96625	7519
Ministry of Agriculture and Forestry	78623	4427.25
Archives New Zealand	72640.25	77
Statistics New Zealand	66164.6	473.8
Ministry of Health	65685.2	5996.2
New Zealand Customs Service	62943.2	-163.8
Land Information New Zealand	60741.6	-15511
Ministry of Economic Development	57515	-4094.4
Ministry of Fisheries	37446.4	1161
Department of Building and Housing	35173.6	5039.8
Parliamentary Service	33797	2046.6
State Services Commission	25189	-2155.8
Ministry of Science and Innovation	18338	344
Ministry of Transport	12749.6	1496.2
Ministry for Culture and Heritage	4452.8	1277.8

3.1.2 Data Preprocessing

The data files downloaded are in Microsoft Excel format. For convenience for data preprocessing, we convert the Excel file to intermediate CSV files. The CSV files are then imported into a temporary SQL table `nzgovt` by SQL statements. Listing 3.1 shows an example SQL statement for loading the 2008 budget into the system. Budgets of other fiscal years are imported using the same SQL statement by replacing the CSV filename correspondingly.

Listing 3.1: *SQL Statement to Import CSV*

```
1 LOAD DATA INFILE '/b08-dept-ffs-data.csv' REPLACE INTO TABLE `passage`.`nzgovt` FIELDS TERMINATED BY ','
      ENCLOSED BY '"' LINES TERMINATED BY '\n' IGNORE 1 LINES;
2 LOAD DATA INFILE '/b09-dept-ffs-data.csv'  REPLACE INTO TABLE `passage`.`nzgovt` FIELDS TERMINATED BY ','
      ENCLOSED BY '"' LINES TERMINATED BY '\n' IGNORE 1 LINES;
3 ...
```

In our case, the raw data is 23 selected New Zealandgovernment department budgets for 5 fiscal years from 2008 to 2012. We calculate the common components from the data stored in SQL table `nzgovt`. The calculation is done by a SQL stored procedure `submitNZGovtBudget` which is called by Java code as shown in Listing 3.2. This practice is repeated until all departmental budgets for all fiscal years are imported into the system. The steps of the data preprocessing is illustrated in Figure 3.1.

Listing 3.2: *Java Code to Execute SQL Stored Procedure* `submitNZGovtBudget`

```
1  ...
2  int[] years = new int[]{2007, 2008, 2009, 2010, 2011, 2012, 2013};
3  String[] types = new String[]{"Actual", "Forecast"};
4  for (String company : companies) {
5        for (int year : years) {
6             for (String type : types) {
7                sqlClient.callProcedure("submitNZGovtBudget", company, year, type);
8             }
9        }
10 }
11 ...
```

Following the methodology in Section 2.2.1, the financial variables ΔX_1 to ΔX_{19} of budgets are calculated from the common components stored in SQL table `budgets`. The Java code used for this operation is shown in Listing 3.3. 6 criteria in Table 2.1 are used to label viable (class: 1) or inviable (class: 0) budgets. As a result, 6 training sets are generated and used for testing with 8 different algorithms using leave-one-out cross validation as described in the next Section 3.2.

Listing 3.3: *Java Code to Execute* `getTrainingSets`

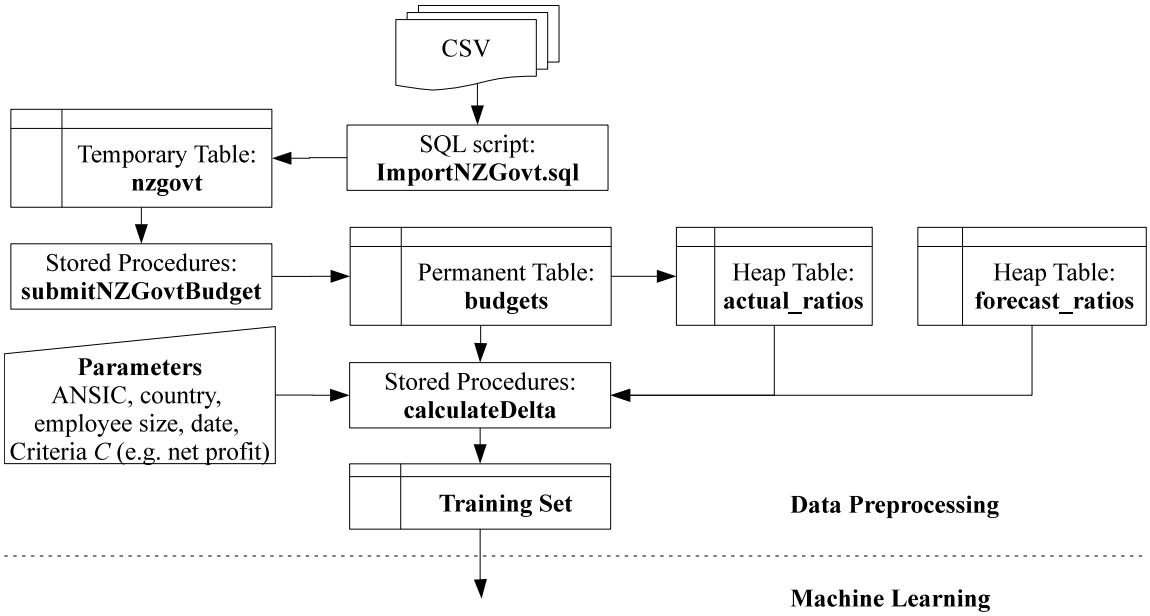

Figure 3.1: *Data Preprocessing (to be contined in Figure 3.2)*

```
...
tests.addAll(Arrays.asList(
            "sales - cog - oper_cost - interest - tax", //net income
            "cash_flow",
            "1.2 * work_cap / total_asset + 1.4 * ret_earning / total_asset + 3.3 * ebit / total_asset + 0.6 *
                 market_equity / total_liab + 1.0 * sales / total_asset", //z-score
            "net_income / sales", //ROA
            "sales / total_asset", //profit margin
            "net_income / equity" //ROE
));
String anzsic = "OO211"; //government
Periodicity periodicity = Periodicity.ANNUALLY;
trainingSets = getTrainingSets(anzsic, periodicity, tests.toArray());
...
```

3.2 Experimental Setup

Weka (Hall et al., 2009) is a open-source data mining application developed by University of Waikato. 8 classification algorithms from Weka E1 to E8 are employed to test on the 6 training sets generated in the previous section. The 8 classification algorithms are accessed through the Weka API as listed in Table 3.3. The experiment steps are illustrated in Figure 3.2.

Giving the full class path of a classification algorithms from Weka, an `Engine` object is instantiated as shown in Listing 3.4. By providing a training set generated previously,

Table 3.3: *Prediction Algorithms*

	Engine	Java Class Path
E1	Weka 3.6.4-LibSVM	weka.classifiers.functions.LibSVM
E2	Weka 3.6.4-SMO	weka.classifiers.functions.SMO
E3	Weka 3.6.4-MultilayerPerceptron	weka.classifiers.functions.MultilayerPerceptron
E4	Weka 3.6.4-RBFNetwork	weka.classifiers.functions.RBFNetwork
E5	Weka 3.6.4-NaiveBayes	weka.classifiers.bayes.NaiveBayes
E6	Weka 3.6.4-ADTree	weka.classifiers.trees.ADTree
E7	Weka 3.6.4-BFTree	weka.classifiers.trees.BFTree
E8	Weka 3.6.4-IBk	weka.classifiers.lazy.IBk

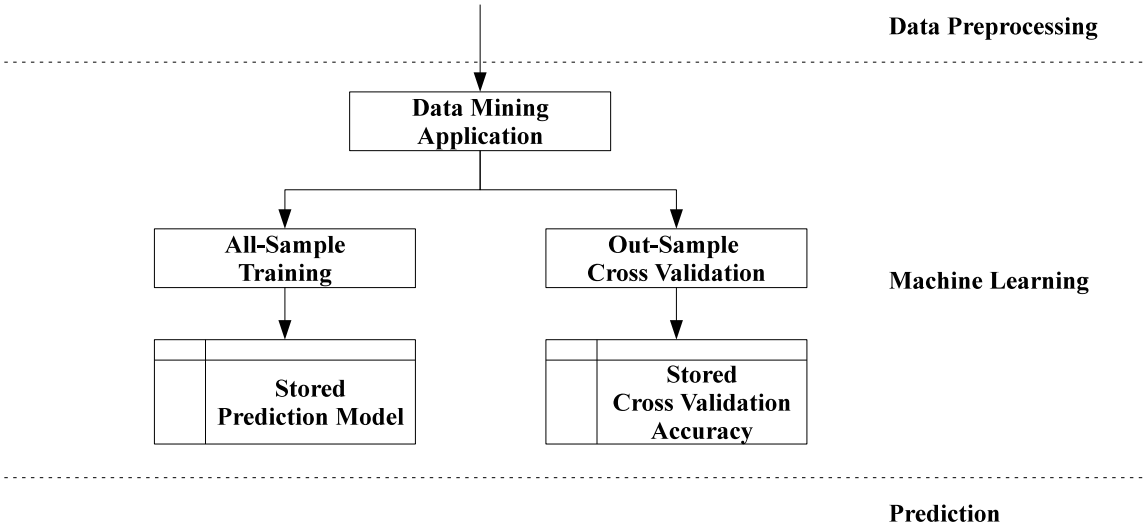

Figure 3.2: *Machine Learning (to be contined in Figure 3.3)*

the `Engine` object performs cross-validations by calling the method `crossValidate`. This operation is repeated until all the 6 training set are performed cross-validation with all the 8 classification algorithms as shown in Listing 3.5.

Listing 3.4: *Java Code to Instantiate* ***`Engine`***

```
...
engines.addAll(Arrays.asList(new Engine[]
{
new Weka() {
        @Override
        public Class<? extends Classifier> getClassifierType()
        {
                return weka.classifiers.functions.LibSVM.class; // E1
        }
},
// same as E2 to E8
}
}));
...
```

Listing 3.5: *Java Code to Perform Cross-Validation*

```
...
for (int i = 0; i < engines.size(); i++) {
        Engine engine = engines.get(i);
        for (int k = 0; k < trainingSets.size(); k++) {
                TrainingSet trainingSet = trainingSets.get(k);
                // ...
                        CrossValidation cv = engine.crossValidate(trainingSet);
         }
}
...
```

3.3 Performance Evaluation

Performing 8 classification algorithms E1 to E8 on 6 training sets T1 to T6, 48 cross validation are completed. The contingency tables of the cross-validations are listed in Table 3.4. The diagonal numbers TP (True Positive) $-TN$ (True Negative) of each of the cells are bolded, which represent the correctly predicted occurrences and these are our focal interest. The prediction is better if more numbers concentrate in the diagonal of the contingency table. However, at the same time, the FN (False Negative) should be kept low to minimise the error (E) (refer to Equation 2.9). It is because wrongly predicted positives (FP) gives a false green light to a forecast budget, which is dangerous if the budget eventually cannot be achieved. Whereas FP (False Positive) is relatively safer, which may cause no harm but result in a surprising outperformed budget. Therefore, to figure out a good classifier, we first look at the diagonal of the contingency table and then the error E.

Table 3.4: *Contingency Tables*

TP FP / FN TN	T1		T2		T3		T4		T5		T6	
Weka 3.6.4-LibSVM	**48**	12	**35**	25	**34**	19	**49**	11	**0**	0	**48**	12
	0	**0**	0	**0**	1	**6**	0	**0**	19	**41**	0	**0**
Weka 3.6.4-SMO	**47**	12	**35**	25	**34**	17	**48**	11	**0**	0	**47**	12
	1	**0**	0	**0**	1	**8**	1	**0**	19	**41**	1	**0**
Weka 3.6.4-MultilayerPerceptron	**44**	7	**18**	13	**27**	12	**45**	6	**5**	7	**44**	7
	4	**5**	17	**12**	8	**13**	4	**5**	14	**34**	4	**5**
Weka 3.6.4-RBFNetwork	**41**	12	**24**	18	**30**	12	**48**	9	**6**	5	**41**	12
	7	**0**	11	**7**	5	**13**	1	**2**	13	**36**	7	**0**
Weka 3.6.4-NaiveBayes	**35**	4	**16**	15	**32**	14	**36**	3	**7**	4	**35**	4
	13	**8**	19	**10**	3	**11**	13	**8**	12	**37**	13	**8**
Weka 3.6.4-ADTree	**41**	10	**23**	10	**27**	10	**42**	7	**8**	7	**41**	10
	7	**2**	12	**15**	8	**15**	7	**4**	11	**34**	7	**2**
Weka 3.6.4-BFTree	**41**	11	**30**	22	**31**	12	**48**	8	**4**	5	**42**	11
	7	**1**	5	**3**	4	**13**	1	**3**	15	**36**	6	**1**
Weka 3.6.4-IBk	**39**	8	**22**	15	**29**	12	**42**	7	**7**	12	**39**	8
	9	**4**	13	**10**	6	**13**	7	**4**	12	**29**	9	**4**

Interestingly, class imbalance is found in most of the training sets. This characteristic is due to the different numbers between two classes in a training dataset. It may undermine the accuracy of some prediction models. We found that this problem is suffering LibSVM (E1) and SMO (E2), two of the variances of SVM. It may suggest that a class imbalance capable version of SVM is required. Apart from the two SVMs, the top three algorithms with highest overall accuracy A $(TP + TN)$ and minimal error E are highlighted in Table 3.5. They are MultilayerPerceptron (E3) NaiveBayes (E5) and ADTree (E6).

Table 3.5: *Average Precision*

Engine	Accuracy	Precision	Recall	Error	F	NP
Weka 3.6.4-LibSVM	0.725		0.829	0.793		
Weka 3.6.4-SMO	0.722		0.818	0.78		
Weka 3.6.4-MultilayerPerceptron	**0.714**	**0.716**	**0.717**	**0.48**	**0.713**	**0.568**
Weka 3.6.4-RBFNetwork	0.689	0.703	0.758	0.69	0.722	0.419
Weka 3.6.4-NaiveBayes	**0.675**	**0.761**	**0.655**	**0.366**	**0.695**	**0.505**
Weka 3.6.4-ADTree	**0.706**	**0.738**	**0.736**	**0.546**	**0.735**	**0.462**
Weka 3.6.4-BFTree	0.714	0.704	0.784	0.66	0.73	0.488
Weka 3.6.4-IBk	0.672	0.698	0.718	0.557	0.707	0.468

As far as computing resources is concerned, the top three algorithms with shortest average processing time are shown in Table 3.6. They are NaiveBayes (E5), ADTree (E6) and IBK (E8). Therefore, by the performance evaluations of the two aspects,

NaiveBayes (E5) and ADTree (E6) are the final shortlisted candidates for prediction in Section 3.4.

Table 3.6: *Processing Time (in second)*

Engine	T1	T2	T3	T4	T5	T6	Mean
Weka 3.6.4-LibSVM	0.427	0.232	0.196	0.193	0.234	0.114	0.233
Weka 3.6.4-SMO	0.845	0.596	0.609	0.721	0.651	0.677	0.683
Weka 3.6.4-MultilayerPerceptron	8.608	3.61	3.581	3.535	3.536	3.564	4.406
Weka 3.6.4-RBFNetwork	1.28	0.3	0.246	0.236	0.218	0.171	0.409
Weka 3.6.4-NaiveBayes	**0.147**	**0.058**	**0.044**	**0.036**	**0.039**	**0.035**	**0.06**
Weka 3.6.4-ADTree	**0.159**	**0.13**	**0.126**	**0.117**	**0.118**	**0.12**	**0.128**
Weka 3.6.4-BFTree	0.461	0.314	0.19	0.138	0.187	0.16	0.242
Weka 3.6.4-IBk	**0.035**	**0.035**	**0.035**	**0.034**	**0.036**	**0.034**	**0.035**

3.4 Experimental Results

The classification model and cross validated accuracy are saved on disk as illustrated in Figure 3.2. Thus a new instance of Java class `Engine` can be constructed from the saved file without training. The `predict` method of the classification model is called to classify a new forecast budget as shown in Listing 3.6. Given a forecast budget and the criteria of test, for example net profit, the classification model returns an output either '1' or '0'. An output '1' means the criteria (e.g. net profit) of the forecast budget is more likely to be achievable. Whereas, an output '0' implies an inviable or unrealistic budget. In addition, an expected precision of this classification is given. The classification model also returns the positive precision P and the negative precision NP (see Section 2.3.5) for output '1' and '0' respectively as illustrated in Figure 3.3.

Listing 3.6: *Java Code for classification*

```
1 ...
2 String date = "2012-06-30";
3 List<InputVector> predictionSet = getInputSet(date, date, Periodicity.ANNUALLY);
4 Engine predictionEngine = GeneralUtil.newInstance(engine.getClass());
5 predictionEngine.loadModel(modelFile);
6 Object[] outputs = predictionEngine.predict(predictionSet);
7 ...
```

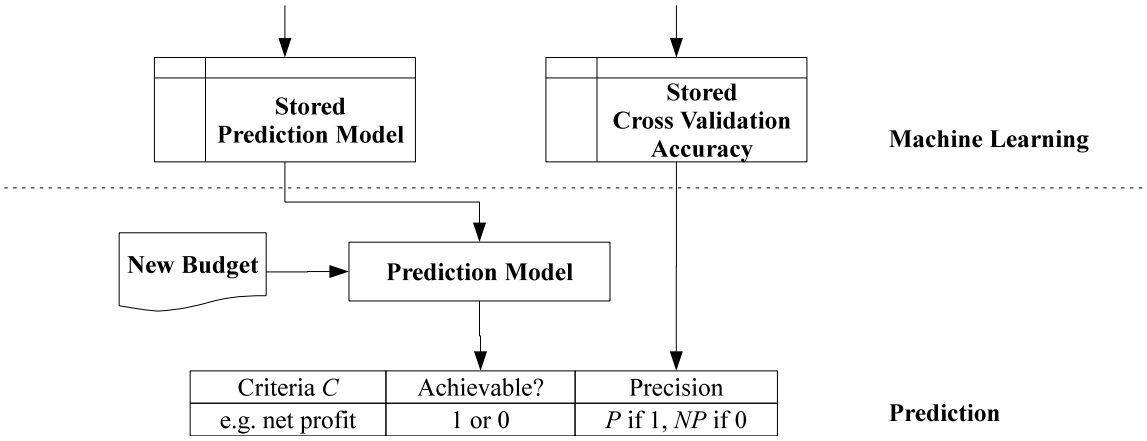

Figure 3.3: *Prediction*

3.4.1 Budget 2011/2012

The New ZealandBudget 2011/2012 is used to test the shortlisted algorithms Naive-Bayes (E5) and ADTree (E6). Each of them are trained by all budgets from fiscal years 2008/2009 to 2010/2011 using the procedures mentioned previously. The cross validation precision of training sets T1 to T6 for the algorithms are shown in Table 3.7 and 3.8 respectively. The budget classification results of the departments are shown in Table 3.9 and 3.10.

Table 3.7: *Prediction Accuracy for NaiveBayes*

	Test	Accuracy	Precision	Recall	Error	F-measure	N. Precision
T1	Net Profit	0.717	0.897	0.729	0.333	0.805	0.381
T2	Cash Flow	0.433	0.516	0.457	0.6	0.485	0.345
T3	z-score	0.717	0.696	0.914	0.56	0.79	0.786
T4	Profit Margin	0.733	0.923	0.735	0.273	0.818	0.381
T5	DuPont ROA	0.733	0.636	0.368	0.098	0.467	0.755
T6	Return on Equity	0.717	0.897	0.729	0.333	0.805	0.381

Table 3.8: *Prediction Accuracy for ADTree*

	Test	Accuracy	Precision	Recall	Error	F-measure	N. Precision
T1	Net Profit	0.717	0.804	0.854	0.833	0.828	0.222
T2	Cash Flow	0.633	0.697	0.657	0.4	0.676	0.556
T3	z-score	0.7	0.73	0.771	0.4	0.75	0.652
T4	Profit Margin	0.767	0.857	0.857	0.636	0.857	0.364
T5	DuPont ROA	0.7	0.533	0.421	0.171	0.471	0.756
T6	Return on Equity	0.717	0.804	0.854	0.833	0.828	0.222

Generally, both algorithms, NaiveBayes (E5) and ADTree (E6), have reasonably

Table 3.9: *Prediction for Budget 2011/2012 Using NaiveBayes*

Company	T1	T2	T3	T4	T5	T6
Ministry of Education	0(38%)	0(34%)	1(70%)	0(38%)	0(76%)	0(38%)
Department of Corrections	0(38%)	0(34%)	1(70%)	0(38%)	0(76%)	0(38%)
Ministry of Justice	0(38%)	0(34%)	1(70%)	0(38%)	0(76%)	0(38%)
Department of Conservation	0(38%)	0(34%)	0(79%)	0(38%)	0(76%)	0(38%)
Ministry of Foreign Affairs and Trade	0(38%)	0(34%)	1(70%)	0(38%)	0(76%)	0(38%)
Department of Internal Affairs	1(90%)	0(34%)	1(70%)	1(92%)	0(76%)	1(90%)
Department of Labour	0(38%)	1(52%)	0(79%)	0(38%)	1(64%)	0(38%)
Statistics New Zealand	1(90%)	1(52%)	1(70%)	1(92%)	0(76%)	1(90%)
Ministry of Health	0(38%)	0(34%)	0(79%)	1(92%)	1(64%)	0(38%)
New Zealand Customs Service	1(90%)	0(34%)	0(79%)	1(92%)	0(76%)	1(90%)
Land Information New Zealand	1(90%)	1(52%)	1(70%)	1(92%)	0(76%)	1(90%)
Ministry of Economic Development	0(38%)	1(52%)	0(79%)	0(38%)	1(64%)	0(38%)
Ministry of Fisheries	0(38%)	0(34%)	1(70%)	0(38%)	0(76%)	0(38%)
Department of Building and Housing	1(90%)	1(52%)	0(79%)	1(92%)	0(76%)	1(90%)
Parliamentary Service	0(38%)	0(34%)	1(70%)	0(38%)	1(64%)	0(38%)
State Services Commission	0(38%)	0(34%)	0(79%)	0(38%)	1(64%)	0(38%)
Ministry of Science and Innovation	1(90%)	0(34%)	1(70%)	1(92%)	0(76%)	1(90%)
Ministry of Transport	1(90%)	0(34%)	0(79%)	1(92%)	0(76%)	1(90%)
Ministry for Culture and Heritage	1(90%)	1(52%)	0(79%)	1(92%)	0(76%)	1(90%)

Table 3.10: *Prediction for Budget 2011/2012 Using ADTree*

Company	T1	T2	T3	T4	T5	T6
Ministry of Education	1(80%)	1(70%)	0(65%)	1(86%)	0(76%)	1(80%)
Department of Corrections	1(80%)	0(56%)	1(73%)	1(86%)	1(53%)	1(80%)
Ministry of Justice	1(80%)	0(56%)	0(65%)	0(36%)	0(76%)	1(80%)
Department of Conservation	1(80%)	0(56%)	0(65%)	1(86%)	1(53%)	1(80%)
Ministry of Foreign Affairs and Trade	1(80%)	0(56%)	1(73%)	1(86%)	0(76%)	1(80%)
Department of Internal Affairs	1(80%)	0(56%)	0(65%)	1(86%)	0(76%)	1(80%)
Department of Labour	1(80%)	1(70%)	0(65%)	1(86%)	0(76%)	1(80%)
Statistics New Zealand	1(80%)	0(56%)	1(73%)	1(86%)	0(76%)	1(80%)
Ministry of Health	1(80%)	0(56%)	0(65%)	1(86%)	0(76%)	1(80%)
New Zealand Customs Service	1(80%)	0(56%)	0(65%)	1(86%)	0(76%)	1(80%)
Land Information New Zealand	1(80%)	1(70%)	0(65%)	0(36%)	0(76%)	1(80%)
Ministry of Economic Development	1(80%)	1(70%)	0(65%)	1(86%)	1(53%)	1(80%)
Ministry of Fisheries	0(22%)	1(70%)	0(65%)	0(36%)	1(53%)	0(22%)
Department of Building and Housing	1(80%)	0(56%)	0(65%)	1(86%)	0(76%)	1(80%)
Parliamentary Service	1(80%)	1(70%)	1(73%)	1(86%)	1(53%)	1(80%)
State Services Commission	1(80%)	0(56%)	1(73%)	1(86%)	0(76%)	1(80%)
Ministry of Science and Innovation	1(80%)	1(70%)	0(65%)	1(86%)	0(76%)	1(80%)
Ministry of Transport	0(22%)	0(56%)	0(65%)	0(36%)	0(76%)	0(22%)
Ministry for Culture and Heritage	1(80%)	1(70%)	0(65%)	1(86%)	0(76%)	1(80%)

good positive precision, all of them are well above 50% and have an average of 76% and 74% respectively. They have relatively low percentage of E (37% and 55%), however the negative precision are only marginally half (51% and 46%). Although there is no clear winner in this experiment, the lowest error one, i.e. NaiveBayes (E5), is suggested.

Chapter 4

Conclusions

4.1 Solution to Problem-solving

With the aid of modern budgeting software, SME owners with limited accounting and financial knowledge can create budgets without the hassle of managing error-prone spreadsheets. However, these budgeting softwares often provide no analysis or guidelines on how these budgets are likely to be viable.

Previous researches show that financial variables can predict future events of a business. In this research, we investigate an SME's historical variation between the forecast and actual data on financial variables. For a specific SME industry, we discover its characterised change pattern, and use the knowledge to label every historical forecast as a viable or inviable budget. To alert SME owners to budget inaccuracy and their risks, a classification model is developed for the accuracy inference of SME business budgeting and planning.

As the research outcome, we develop on top of Passage Forecast 5 a data mining software prototype for SME business intelligent budgeting planning. With the aid of open-sourced database and software libraries, the prototype consists of an SQL database for storing Forecast 5 financial variables data, and an API for its integration to Forecast 5. Through system integration, the empowered Forecast 5 is able to trigger an alarm whenever a risk of inaccuracy occurs in the budget setting of an SME. In experiment, we use 23 departments of the New Zealandgovernment as an example of SME, and build the SME intelligent budgeting model on the collected

data. We test the developed software prototype by a leave-one-out cross-validation approach. The result shows that the proposed intelligent budgeting has an overall 70.5% classification accuracy.

4.2 Contribution of the Research

Using the prototype software, the experiment demonstrates the existence of the change pattern of financial variables for a certain SME industry group. A budget inaccuracy alarm system is developed. The system classifies a budget whether viable or inviable. As a result, SME businesses or non-profit organisations can make better decisions by improving forward looking financial analysis. This allows them to immediately develop contingency measures, revise the policy and get the business back on the right track. This has been a long-desired function needed by a business or a bank. Importantly, this opens the door for further investigation for industry specified combination of financial variables for other groups of SME.

4.3 Limitation and Future Development

It is worth noting that sufficient forecast data collection is essential for discovering the financial variable change patterns for a certain SME industry group. However, due to the ethical and confidential issues of company, the data collection in practice is extremely difficult and is often a long process. Thus, insufficient data is a major limitation of the presented research in terms of being able to be used widely in different industries.

On the other hand, when new data is presented, the proposed model needs to be reconstructed from scratch. Thus, how to train the prediction model backwards and forwards incrementally remains a challenging problem and is left for future development.

Glossary

AI artificial intelligence. 5

ANZSIC Australian and New Zealand Standard Industrial Classification. 1, 15

API application programming interface. ii, 11, 13, 25, 33

DMLI Decentralized Machine Learning Intelligence. iii, 1

DuPont DuPont Analysis is developed by F. Donaldson Brown, a staff of DuPont Corporation in 1960's. i, 2, 5–7, 10

FL financial leverage. 5, 6, 11

GA integrating generic algorithm. 7

GAAP generally accepted accounting principles. 3

GNU General Public License A free, copyleft license for software and other kinds of works. 11

IBDR integrated binary discriminant rule. 7

LVQ learning vector quantisation. 7

MDA multiple discriminate analyse. 7

MSI Ministry of Science and Innovation. iii

NN neural network. 7, 42

PM profit margin. 5, 6, 11

ROA return on asset. 5, 6, 10, 11

ROE return on equity. 5, 6, 10

SME small and medium-sized enterprise. i–iii, 1, 2, 4, 7, 10, 22, 33, 34

SQL structured query language. ii, 11–18, 33

SVM support vector machine. 7, 27, 42

Weka Data Mining Software in Java. Weka is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from your own Java code. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes. Weka is open source software issued under the GNU General Public License. More information at www.cs.waikato.ac.nz. 13, 25

References

Altman, E. I. (1968). Financial ratio, discriminant analysis, and the prediction of corporate bankruptcy. *Journal of Finance*, *23*, 589-609.

Beaver, W. (1966). Financial ratios as predictors of failure. empirical research in accounting: Selected studies. *Journal of Accounting Research*, *4*, 71-111.

Beaver, W., McNichols, M. & Rhie, J. (2005). Have financial statements become less informative? evidence from the ability of financial ratios to predict bankruptcy. *Review of Accounting Studies*, *10*(1), 93-122.

Blum, M. (1974). Failing company discriminant analysis. *Journal of Accounting Research*, *12*(1), 1-25.

Chang, C.-C. & Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, *2*, 27:1–27:27. (Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`)

Collier, H. W., McGowan, C. B. & Muhammad, J. (2010). Evaluating the impact of a rapidly changing economic environment on bank financial performance using the DuPont system of financial analysis. *Asia Pacific Journal of Finance and Banking Research*, *4*(4), 25-35.

Deakin, E. B. (1972). A discriminant analysis of predictors of business failure. *Journal of Accounting Research*, *10*(1), 167-179.

Ding, Y., Song, X. & Zen, Y. (2008). Forecasting financial condition of chinese listed companies based on support vector machine. *Expert Systems with Applications*, *34*(4), 3081-3089.

Estes, J. & Savich, R. S. (2011). A comparison of financial analysis software for use in financial planning for small businesses. *Journal of Finance Service Professionals*.

Fight, A. (2006). *Cash flow forecasting*. Burlington, MA: Butterworth-Heinemann.

Gallizo, J., Gargallo, P. & Salvador, M. (2008). Multivariate partial adjustment of financial ratios: a Bayesian hierarchical approach. *Journal of applied econometrics*, *23*(1), 43-64.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. & Witten, I. H. (2009). The WEKA data mining software: an update. *SIGKDD Explor. Newsl.*, *11*(1), 10-18. Available from `http://doi.acm.org/10.1145/1656274.1656278`

Hua, Z., Wang, Y., Xu, X., Zhang, B. & Liang, L. (2007). Predicting corporate financial distress based on integration of support vector machine and logistic regression. *Expert Systems with Applications*, *33*(2), 434-440.

Li, H. & Sun, J. (2009). Predicting financial failure using multiple case-based reasoning combine with support vector machine. *Expert Systems with Applications*, *36*(6), 10085-10096.

Lin, F., Liang, D. & Chen, E. (2011). Financial ratio selection for business crisis prediction. *Expert Systems with Applications*, *38*, 15094–15102.

Martens, D., Bruynseels, L., Baesens, B., Willekens, M. & Vanthienen, J. (2008). Predicting going concern opinion with data mining. *Decision Support Systems*, *45*(4), 765-777.

Min, J. & Lee, Y. (2005). Bankruptcy prediction using support vector machine with optimal choice of kernel function parameters. *Expert Systems with Applications*, *28*(4), 603-614.

Min, S., Lee, J. & Han, I. (2006). Hybrid genetic algorithms and support vector machines for bankruptcy prediction. *Expert Systems with Applications*, *31*(3), 652-660.

Ministry of Economic Development. (2010). *SMEs in New Zealand: Structure and dynamics* (Tech. Rep.). The New Zealand Government.

Narayanan, L. (2010). How DuPont analysis reveals return on equity ratio. *Managing credit, receivables and collections*, 12-13.

Nissim, S., D. Penman. (2001). Ratio analysis and equity valuation: From research to practice. *Review of Accounting Studies*, *6*(1), 109-154.

Ohlson, J. A. (1980). Financial ratios and the probabilistic prediction of bankruptcy. *Journal of Accounting Research*, *18*(1), 109-131.

Ozawa, S., Pang, S. & Kasabov, N. (2008, june). Incremental learning of chunk data for online pattern classification systems. *Neural Networks, IEEE Transactions on*, *19*(6), 1061 -1074.

Pogue, M. (2008). *Business failure: Prediction and prevention.* London, UK: ACCA.

Priester, C. & Wang, J. (2009). *Financial strategies for the manager.* Springer Verlag.

Shin, K., Lee, T. & Kim, H. (2005). An application of support vector machines in bankruptcy prediction model. *Expert Systems with Applications*, *28*(1), 127-135.

Soliman, M. (2008). The use of DuPont analysis by market participants. *The Accounting Review*, *83*(3), 823-853.

Soliman, M. & Lundholm, R. (2003). *Using industry-adjusted DuPont analysis to predict future profitability and returns.* University of Michigan. Available from `http://books.google.com/books?id=dwkeAQAAMAAJ`

Statistics New Zealand. (2010). *Business operations survey: 2010* (Tech. Rep.). Statistics New Zealand.

Sun, J. & Li, H. (2009). Financial distress prediction based on serial combination of multiple classifiers. *Expert Systems with Applications*, *36*(4), 8659–8666.

The Heritage Foundation and The Wall Street Journal. (2011). *Highlights of the 2011 index of economic freedom: Promoting economic opportunity and prosperity* (Tech. Rep.). The Heritage Foundation and The Wall Street Journal.

VantagePoint. (2011). *Growth guide: The business of growth* (Tech. Rep.). VantagePoint.

Zmijewski, M. E. (1984). Methodological issues related to the estimation of financial distress prediction models. *Journal of Accounting Research*, *22*, 59-82.

Appendices

Appendix A

Source Codes

SQL Scripts

Listing A.1: *createTables.sql*

```
CREATE SCHEMA IF NOT EXISTS `passage`;
CREATE USER 'passage'@'localhost';
GRANT ALL ON passage.* TO passage@localhost;
DROP TABLE IF EXISTS `passage`.`companies`;
CREATE  TABLE IF NOT EXISTS `passage`.`companies` (
  `company_id` INT NOT NULL AUTO_INCREMENT,
  `company_name` VARCHAR(100) NOT NULL ,
  `anzsic` VARCHAR(20) NOT NULL ,
  `country` VARCHAR(50) NULL ,
  `size` INT NULL ,
  PRIMARY KEY (`company_id`),
  UNIQUE INDEX `company_name_UNIQUE` (`company_name` ASC)
)
ENGINE = InnoDB;
DROP TABLE IF EXISTS `passage`.`budgets`;
CREATE  TABLE IF NOT EXISTS `passage`.`budgets` (
  `budget_id` INT NOT NULL AUTO_INCREMENT,
  `company_id` INT NOT NULL ,
  `periodicity` DECIMAL(3,1) NOT NULL ,
  `end_date` DATE NOT NULL ,
  `actual` TINYINT(1) NOT NULL ,
  `cash_eq` DOUBLE NULL ,
  `cur_liab` DOUBLE NULL ,
  `non_cur_liab` DOUBLE NULL ,
  `cur_debt` DOUBLE NULL ,
  `non_cur_debt` DOUBLE NULL ,
  `cur_asset` DOUBLE NULL ,
  `non_cur_asset` DOUBLE NULL ,
  `inventory` DOUBLE NULL ,
  `market_equity` DOUBLE NULL ,
  `equity` DOUBLE NULL ,
  `ret_earning` DOUBLE NULL ,
  `cash_flow` DOUBLE NULL ,
  `sales` DOUBLE NULL ,
  `cog` DOUBLE NULL ,
  `oper_cost` DOUBLE NULL ,
  `interest` DOUBLE NULL ,
  `tax` DOUBLE NULL ,
```

```
  PRIMARY KEY (`company_id`, `periodicity`, `end_date`, `actual`),
  UNIQUE INDEX `budget_id_UNIQUE` (`budget_id` ASC)
)
ENGINE = InnoDB;
DROP TABLE IF EXISTS`passage`.`nzgovt`;
CREATE  TABLE IF NOT EXISTS `passage`.`nzgovt` (
  `department` VARCHAR(100) NOT NULL ,
  `statement` VARCHAR(100) NOT NULL ,
  `line_name` VARCHAR(100) NOT NULL ,
  `line_order` INT NOT NULL ,
  `year` INT NOT NULL ,
  `amount_type` VARCHAR(50) NOT NULL ,
  `amount` DOUBLE NOT NULL ,
  PRIMARY KEY (`department`, `statement`, `line_name`, `year`, `amount_type`)
)
ENGINE = InnoDB;
```

Listing A.2: *createProcedures.sql*

```
DELIMITER //
DROP PROCEDURE IF EXISTS `calculateDeltas`//
CREATE PROCEDURE `calculateDeltas` (company VARCHAR(100), anzsic VARCHAR(20), country VARCHAR(3), size_from
     VARCHAR(6), size_to VARCHAR(6), end_date_from VARCHAR(10), end_date_to VARCHAR(10), periodicity DOUBLE,
     test VARCHAR(500))
BEGIN
SET @actual = 0;
SET @company_cond  = IF (company = '' OR company = '*', '', CONCAT(' AND (company_id = "', company, '" OR
     company_name = "', company, '")'));
SET @anzsic_cond = IF (anzsic = '' OR anzsic = '*', '', CONCAT(' AND anzsic = "', anzsic, '"'));
SET @country_cond = IF (country = '' OR country = '*', '', CONCAT(' AND country = "', country, '"'));
SET @size_from_cond = IF (size_from = '' OR size_from = '*', '', CONCAT(' AND size >= ', size_from));
SET @size_to_cond = IF (size_to = '' OR size_to = '*', '', CONCAT(' AND size <= ', size_to));
SET @from_cond =  IF (end_date_from = '' OR end_date_from = '*', '', CONCAT(' AND end_date >= "', LAST_DAY(
     DATE_SUB(end_date_from, INTERVAL periodicity MONTH)), '"'));
SET @to_cond = IF (end_date_to = '' OR end_date_to = '*', '', CONCAT(' AND end_date <= "', end_date_to, '"'));
set @criteria = IF (test = '', 'NULL',
                IF (test = '1', 'net_income',
                IF (test = '2', 'cash_flow',
                IF (test = '3', '1.2 * work_cap / total_asset
                             + 1.4 * ret_earning / total_asset
                             + 3.3 * ebit / total_asset
                             + 0.6 * market_equity / total_liab
                             + 1.0 * sales / total_asset',
                IF (test = '4', 'net_income / sales',
                IF (test = '5', 'sales / total_asset',
                IF (test = '6', 'net_income / equity',
                test)))))));
REPEAT
  SET @tab = CONCAT('`passage`.`', IF(@actual, 'actual', 'forecast'), '_ratios`');
  SET @s1 = CONCAT('DROP TABLE IF EXISTS ', @tab);
  PREPARE stmt FROM @s1;
  EXECUTE stmt;
  SET @s2 = CONCAT(
  ' CREATE TABLE ', @tab, ' ENGINE = MEMORY',
  ' SELECT
      budget_id,
      company_id,
      anzsic,
      country,
      company_name,
      periodicity,
      end_date,
      actual,
      cash_eq / cur_liab AS x1,
      cash_flow / total_debt AS x2,
      cash_flow / total_asset AS x3,
      cash_flow / sales AS x4,
      total_liab / total_asset AS x5,
      work_cap / total_asset AS x6,
```

```
      market_equity / total_asset AS x7,
      cur_asset / total_asset AS x8,
      quick_asset / total_asset AS x9,
      net_sales / total_asset AS x10,
      cur_debt / sales AS x11,
      quick_asset / sales AS x12,
      work_cap / sales AS x13,
      net_income / total_asset AS x14,
      ret_earning / total_asset AS x15,
      ebit / total_asset AS x16,
      net_income / sales AS x17,
      sales / total_asset AS x18,
      total_asset / equity AS x19, ',
      @criteria , ' AS criteria ',
  '  FROM
    (SELECT
      *,
      (cur_debt + non_cur_debt) AS total_debt,
      (cur_asset + non_cur_asset) AS total_asset,
      (cur_liab + non_cur_liab) AS total_liab,
      (cur_asset - cur_liab) AS work_cap,
      (cur_asset - inventory) AS quick_asset,
      (sales - cog) AS net_sales,
      (sales - cog - oper_cost) AS ebit,
      (sales - cog - oper_cost - interest - tax) AS net_income
    FROM `passage`.`budgets`
    JOIN `passage`.`companies` USING (company_id)
    WHERE periodicity = ', periodicity,
  ' AND actual = ', @actual,
  @company_cond,
  @anzsic_cond,
  @country_cond,
  @size_from_cond,
  @size_to_cond,
  @from_cond,
  @to_cond,
  ') AS t1'
  );
  PREPARE stmt FROM @s2;
  EXECUTE stmt;
  SET @actual = @actual + 1;
UNTIL @actual > 1 END REPEAT;
SELECT * FROM
(SELECT deltas.*, diff, IF(diff IS NULL, NULL, IF(diff >=0, 1E0, 0E0)) AS class
FROM
  (SELECT
    forecast_ratios.budget_id,
    forecast_ratios.company_id,
    forecast_ratios.anzsic,
    forecast_ratios.country,
    forecast_ratios.company_name,
    forecast_ratios.end_date,
    forecast_ratios.x1 - actual_ratios.x1 AS delta_x1,
    forecast_ratios.x2 - actual_ratios.x2 AS delta_x2,
    forecast_ratios.x3 - actual_ratios.x3 AS delta_x3,
    forecast_ratios.x4 - actual_ratios.x4 AS delta_x4,
    forecast_ratios.x5 - actual_ratios.x5 AS delta_x5,
    forecast_ratios.x6 - actual_ratios.x6 AS delta_x6,
    forecast_ratios.x7 - actual_ratios.x7 AS delta_x7,
    forecast_ratios.x8 - actual_ratios.x8 AS delta_x8,
    forecast_ratios.x9 - actual_ratios.x9 AS delta_x9,
    forecast_ratios.x10 - actual_ratios.x10 AS delta_x10,
    forecast_ratios.x11 - actual_ratios.x11 AS delta_x11,
    forecast_ratios.x12 - actual_ratios.x12 AS delta_x12,
    forecast_ratios.x13 - actual_ratios.x13 AS delta_x13,
    forecast_ratios.x14 - actual_ratios.x14 AS delta_x14,
    forecast_ratios.x15 - actual_ratios.x15 AS delta_x15,
    forecast_ratios.x16 - actual_ratios.x16 AS delta_x16,
    forecast_ratios.x17 - actual_ratios.x17 AS delta_x17,
    forecast_ratios.x18 - actual_ratios.x18 AS delta_x18,
    forecast_ratios.x19 - actual_ratios.x19 AS delta_x19
```

```
    FROM
    (SELECT *, LAST_DAY(DATE_SUB(end_date, INTERVAL periodicity MONTH)) AS prev_end_date
      FROM forecast_ratios
    ) AS forecast_ratios
    LEFT JOIN actual_ratios
    ON (forecast_ratios.prev_end_date = actual_ratios.end_date
    AND forecast_ratios.company_id = actual_ratios.company_id)
    WHERE actual_ratios.end_date IS NOT NULL
  ) AS deltas
JOIN
  (SELECT forecast.budget_id, (actual.criteria - forecast.criteria) AS diff
      FROM
      (SELECT budgets.*,criteria FROM `budgets` JOIN actual_ratios USING (budget_id)) AS actual
      RIGHT JOIN
      (SELECT budgets.*,criteria FROM `budgets` JOIN forecast_ratios USING (budget_id)) AS forecast
      USING (company_id, end_date)
  ) AS criteria_accuracy
USING (budget_id)
ORDER BY company_id, end_date) AS trainingSet
;
END
//
DROP PROCEDURE IF EXISTS `submitNZGovtBudget`//
CREATE PROCEDURE `submitNZGovtBudget` (dept VARCHAR(100), fiscalYear INT, type VARCHAR(50))
BEGIN
SET @anzsic = '00211';
SET @country = 'NZL';
INSERT IGNORE INTO companies (company_name, anzsic, country) VALUES(dept, @anzsic, @country);
SET @company_id = (SELECT company_id FROM companies WHERE company_name = dept);
SET @periodicity = 12;
SET @end_date = CONCAT(fiscalYear, '-06-30');
SET @actual = IF (type LIKE '%ACTUAL', 1, 0);
SET @alt_type = IF (type = 'FORECAST', 'Budget%', '');
SET @count = (SELECT COUNT(amount) FROM `nzgovt` WHERE department = dept AND year = fiscalYear AND (amount_type
     = type OR amount_type LIKE @alt_type));
IF (@count > 0)
THEN
    DROP TABLE IF EXISTS `temp_balance`;
    DROP TABLE IF EXISTS `temp_cash`;
    DROP TABLE IF EXISTS `temp_income`;
    CREATE TABLE `temp_balance` ENGINE = MEMORY SELECT line_name, amount FROM `nzgovt` WHERE department = dept
         AND year = fiscalYear AND (amount_type = type OR amount_type LIKE @alt_type) AND statement LIKE 'Fin%
         Position';
    CREATE TABLE `temp_cash` ENGINE = MEMORY SELECT line_name, amount FROM `nzgovt` WHERE department = dept AND
         year = fiscalYear AND (amount_type = type OR amount_type LIKE @alt_type) AND statement LIKE 'Cash Flow%
         ';
    CREATE TABLE `temp_income` ENGINE = MEMORY SELECT line_name, amount FROM `nzgovt` WHERE department = dept
         AND year = fiscalYear AND (amount_type = type OR amount_type LIKE @alt_type) AND (statement LIKE '%
         Income%' OR statement LIKE 'Fin% Performance');
    SET @cash_eq = (SELECT amount FROM `temp_balance` WHERE line_name = 'Cash and cash equivalents');
    SET @cur_liab = (SELECT amount FROM `temp_balance` WHERE line_name = 'Total Current Liabilities');
    SET @non_cur_liab = (SELECT amount FROM `temp_balance` WHERE line_name = 'Total Current Liabilities');
    SET @cur_debt = (SELECT amount FROM `temp_balance` WHERE line_name = 'Creditors and other payables');
    SET @non_cur_debt = (SELECT amount FROM `temp_balance` WHERE line_name = 'Provisions');
    SET @cur_asset = (SELECT amount FROM `temp_balance` WHERE line_name = 'Total Current Assets');
    SET @non_cur_asset = (SELECT amount FROM `temp_balance` WHERE line_name = 'Total Non-current Assets');
    SET @inventory = (SELECT amount FROM `temp_balance` WHERE line_name = 'Inventories');
    SET @market_equity = 0;
    SET @equity = (SELECT amount FROM `temp_balance` WHERE line_name LIKE 'Total Taxpayers%');
    SET @ret_earning = 0;
    SET @cash_flow = (SELECT amount FROM `temp_cash` WHERE line_name LIKE 'Net Increase%');
    SET @sales = (SELECT amount FROM `temp_income` WHERE line_name = 'Total Income');
    SET @cog = (SELECT amount FROM `temp_income` WHERE line_name = 'Personnel');
    SET @oper_cost = (SELECT amount FROM `temp_income` WHERE line_name = 'Operating')
                   + (SELECT amount FROM `temp_income` WHERE line_name = 'Depreciation and amortisation')
                   + (SELECT amount FROM `temp_income` WHERE line_name = 'Capital charge')
                   + (SELECT amount FROM `temp_income` WHERE line_name = 'Other');
    SET @interest = (SELECT amount FROM `temp_income` WHERE line_name = 'Finance costs');
    SET @tax = 0;
    REPLACE INTO `budgets` (company_id, periodicity, end_date, actual, cash_eq, cur_liab, non_cur_liab, cur_debt
         , non_cur_debt, cur_asset, non_cur_asset, inventory, market_equity, equity, ret_earning, cash_flow,
```

```
        sales, cog, oper_cost, interest, tax)
    VALUES (@company_id, @periodicity, @end_date, @actual, @cash_eq, @cur_liab, @non_cur_liab, @cur_debt,
        @non_cur_debt, @cur_asset, @non_cur_asset, @inventory, @market_equity, @equity, @ret_earning,
        @cash_flow, @sales, @cog, @oper_cost, @interest, @tax);
END IF;
END
//
DELIMITER ;
```

Listing A.3: *importNZGovt.sql*

```
LOAD DATA INFILE '/b08-dept-ffs-data.csv'
REPLACE INTO TABLE `passage`.`nzgovt` FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'
IGNORE 1 LINES;
LOAD DATA INFILE '/b09-dept-ffs-data.csv'
REPLACE INTO TABLE `passage`.`nzgovt` FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'
IGNORE 1 LINES;
LOAD DATA INFILE '/b10-dept-ffs-data.csv'
REPLACE INTO TABLE `passage`.`nzgovt` FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'
IGNORE 1 LINES;
LOAD DATA INFILE '/b11-dept-ffs-data.csv'
REPLACE INTO TABLE `passage`.`nzgovt` FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'
IGNORE 1 LINES;
LOAD DATA INFILE '/b12-dept-ffs-data.csv'
REPLACE INTO TABLE `passage`.`nzgovt` FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'
IGNORE 1 LINES;
```

Java Classes and Libraries

Table A.1: *Java Libraries*

Library	Description	Author	Available at
LIBSVM	Open source SVM (required by Weka)	Chang and Lin (2011)	`http://www.csie.ntu.edu.tw/~cjlin/libsvm`
MySQL JDBC Driver	Open-source MySQL connector	Oracle	`http://dev.mysql.com/downloads/connector/j/`
Neuroph 2.5.1	Open-source NN		`http://neuroph.sourceforge.net`
Weka 3.6.4	Open-source data mining software	University of Waikato	`http://www.cs.waikato.ac.nz/ml/weka/`

Java Package: api

Listing A.4: *BudgetCalculator.java*

```
package api;
import api.budget.Budget;
import api.budget.Periodicity;
import api.company.Company;
import api.engine.InputVector;
import api.engine.TrainingSet;
import api.engine.TrainingVector;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Arrays;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import util.DateUtil;
import util.GeneralUtil;
import util.SqlClient;
import util.SqlUtil;

public class BudgetCalculator {

    public static final String DEFAULT_CLASS_COLUMN = "class";

    public static final String DEFAULT_ATTRIBUTE_PREFIX = "delta_x";
    public static SqlClient sqlClient = new SqlClient();
    public static Integer[] getAttributeNumbers() {
        return new Integer[]{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19};
    }

    public static String[] getAttributeNames() {
        Integer[] numbers = getAttributeNumbers();
        if (numbers == null) {
            return null;
        }
        String[] names = new String[numbers.length];
        for (int i = 0; i < numbers.length; i++) {
            names[i] = DEFAULT_ATTRIBUTE_PREFIX + numbers[i];
        }
        return names;
    }

    public static String[] getAttributeNamesWithClass() {
        return GeneralUtil.join(getAttributeNames(), DEFAULT_CLASS_COLUMN);
    }

    public TrainingSet getTrainingSet(String company, String anzsic, String country, String sizeFrom, String
         sizeTo, String dateFrom, String dateTo, Periodicity periodicity, Object test) throws SQLException,
         ClassNotFoundException {
        log(GeneralUtil.SECTION_BREAK);
        log("Calculating training set...");
        sqlClient.connect();
        Object[][] table = SqlUtil.getTable(
                sqlClient.callProcedure(
                "calculateDeltas", // Usage of `calculateDeltas` (company VARCHAR(100), anzsic VARCHAR(20),
                     country VARCHAR(3), size_from VARCHAR(5), size_to VARCHAR(5), end_date_from VARCHAR(10),
                     end_date_to VARCHAR(10), periodicity DOUBLE, test VARCHAR(500))
                company == null ? "" : company,
                anzsic == null ? "" : anzsic,
                country == null ? "" : country,
                sizeFrom == null ? "" : sizeFrom,
                sizeTo == null ? "" : sizeTo,
                dateFrom == null ? "" : dateFrom,
                dateTo == null ? "" : dateTo,
                periodicity.value,
                test == null ? "" : test),
```

```
62                  getAttributeNamesWithClass());
            if (table == null || table.length == 0) {
64                  throw new UnsupportedOperationException("No data.");
            }
66          List<Object[]> list = new LinkedList<Object[]>(Arrays.asList(table));
            List<TrainingVector> trainingSet = new LinkedList<TrainingVector>();
68          for (Iterator<Object[]> it = list.iterator(); it.hasNext();) {
                Object[] row = it.next();
70              Object[] input = Arrays.copyOf(row, row.length - 1);
                Object output = row[row.length - 1];
72              if (output == null) {
                    it.remove();
74              } else {
                    trainingSet.add(new TrainingVector(input, output));
76              }
            }
78          //display the training set matrix
            log(GeneralUtil.toString(getAttributeNamesWithClass(), list, " | ", 9));
80          return new TrainingSet(trainingSet);
        }
82
        public List<InputVector> getInputSet(String company, String dateFrom, String dateTo, Periodicity periodicity
             ) throws SQLException, ClassNotFoundException {
84          log(GeneralUtil.SECTION_BREAK);
            log("Calculating input set...");
86          sqlClient.connect();
            ResultSet resultSet =
88                  sqlClient.callProcedure(
                    "calculateDeltas", // Usage of `calculateDeltas` (company VARCHAR(100), anzsic VARCHAR(20),
                         country VARCHAR(3), size_from VARCHAR(5), size_to VARCHAR(5), end_date_from VARCHAR(10),
                         end_date_to VARCHAR(10), periodicity DOUBLE, test VARCHAR(500))
90                  company == null ? "" : company, //company
                    "", //anzsic
92                  "", //country
                    "", //size_from
94                  "", //size_to
                    dateFrom == null ? "" : dateFrom, //date_from
96                  dateTo == null ? "" : dateTo, //date_to
                    periodicity.value,
98                  "");
            Object[][] table = SqlUtil.getTable(
100                 resultSet,
                    getAttributeNames());
102         Object[][] budget = SqlUtil.getTable(
                    resultSet,
104                 new String[]{
                        "company_id",
106                     "anzsic",
                        "country",
108                     "company_name",
                        "end_date"
110                 });
            if (table == null || table.length == 0) {
112             throw new UnsupportedOperationException("No data.");
            }
114         List<InputVector> inputSet = new LinkedList<InputVector>();
            for (int i = 0; i < table.length; i++) {
116             InputVector inputVector = new InputVector(table[i]);
                inputVector.setBudget(
118                     new Budget(
                        new Company(budget[i][0].toString(), budget[i][1].toString(), budget[i][2].toString(),
                             budget[i][3].toString()),
120                     periodicity,
                        DateUtil.getCalendarDate(budget[i][4].toString(), "yyyy-MM-dd"),
122                     false));
                inputSet.add(inputVector);
124         }
            //display the prediction set matrix
126         log(GeneralUtil.toString(getAttributeNames(), table, " | ", 9));
            return inputSet;
128     }
```

```
    private void log(Object o) {
130         System.out.println(o);
    }
132 }
```

Listing A.5: *BudgetSubmitter.java*

```
package api;
2 import api.budget.Budget;
import java.sql.ResultSet;
4 import java.sql.SQLException;
import java.util.logging.Level;
6 import java.util.logging.Logger;
import util.DateUtil;
8 import util.GeneralUtil;
import util.SqlClient;
10 import util.SqlUtil;

12 public final class BudgetSubmitter {
    private DataCollector dataCollector;
14     private SqlClient sqlClient = new SqlClient();

16     public BudgetSubmitter(DataCollector dataCollector) {
        this.dataCollector = dataCollector;
18     }

20     public int submitBudget(Budget budget) {
        if (budget == null || !dataCollector.isAvailable(budget)) {
22             return -1;
        }
24         replaceSQL("companies",
                new String[]{"company_id", "company_name", "anzsic", "country", "size"},
26                 new Object[]{
                    budget.getCompany().getId(),
28                     budget.getCompany().getName(),
                    budget.getCompany().getAnzsic(),
30                     budget.getCompany().getCountry(),
                    budget.getCompany().getEmployeeSize()
32                 });
        replaceSQL("budgets",
34                 new String[]{"company_id", "periodicity", "end_date", "actual",
                    "cash_eq", "cur_liab", "non_cur_liab", "cur_debt", "non_cur_debt",
36                     "cur_asset", "non_cur_asset", "inventory", "market_equity", "equity",
                    "ret_earning", "cash_flow", "sales", "cog", "oper_cost", "interest", "tax"},
38                 getBudgetData(budget));
        int budgetId = getBudgetId(budget);
40         return budgetId;
    }
42
    protected Object[] getBudgetData(Budget budget) {
44         Object[] data = new Object[]{
            budget.getCompany().getId(),
46             budget.getPeriodicity().value,
            budget.getEndDateString(),
48             budget.isActual() ? 1 : 0,
            dataCollector.getCashEquivalents(budget),
50             dataCollector.getCurrentLiabilities(budget),
            dataCollector.getNonCurrentLiabilities(budget),
52             dataCollector.getCurrentDebts(budget),
            dataCollector.getNonCurrentDebts(budget),
54             dataCollector.getCurrentAssets(budget),
            dataCollector.getNonCurrentAssets(budget),
56             dataCollector.getInventories(budget),
            dataCollector.getMarketValueEquity(budget),
58             dataCollector.getEquity(budget),
            dataCollector.getRetainedEarnings(budget),
60             dataCollector.getCashFlow(budget),
            dataCollector.getSales(budget),
62             dataCollector.getCostOfGoods(budget),
```

```
            dataCollector.getOperatingCost(budget),
            dataCollector.getInterestCost(budget),
            dataCollector.getTax(budget)
        };
        return data;
    }

    protected void replaceSQL(String table, String[] columns, Object[] values) {
        StringBuilder stat = new StringBuilder();
        try {
            sqlClient.connect();
            stat.append(" REPLACE INTO ");
            stat.append(SqlUtil.quote(table, "`"));
            stat.append(" (");
            stat.append(GeneralUtil.toString(SqlUtil.quote(columns, "`")));
            stat.append(" )");
            stat.append(" VALUES (");
            stat.append(GeneralUtil.toString(SqlUtil.quote(values, "'")));
            stat.append(" );");
            sqlClient.executeUpdate(stat.toString());
            sqlClient.disconnect();
        } catch (SQLException ex) {
            System.out.println(stat.toString());
            Logger.getLogger(BudgetSubmitter.class.getName()).log(Level.SEVERE, null, ex);
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(BudgetSubmitter.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    protected int getBudgetId(Budget budget) {
        int id = -1;
        try {
            sqlClient.connect();
            ResultSet resultSet = sqlClient.executeQuery(
                    "SELECT `budget_id` FROM `budgets`"
                    + " WHERE `company_id` = '" + budget.getCompany().getId() + "'"
                    + " AND `periodicity` = " + budget.getPeriodicity().value
                    + " AND `end_date` = '" + DateUtil.getDateString(budget.getEndDate()) + "'"
                    + " AND `actual` = " + (budget.isActual() ? 1 : 0));
            while (resultSet.next()) {
                id = resultSet.getInt(1);
            }
        } catch (SQLException ex) {
            Logger.getLogger(BudgetSubmitter.class.getName()).log(Level.SEVERE, null, ex);
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(BudgetSubmitter.class.getName()).log(Level.SEVERE, null, ex);
        }
        return id;
    }
}
```

Listing A.6: *DataCollector.java*

```
package api;
import api.budget.Budget;

public interface DataCollector {

    boolean isAvailable(Budget budget);

    double getCashEquivalents(Budget budget);

    double getCurrentLiabilities(Budget budget);

    double getNonCurrentLiabilities(Budget budget);

    double getCurrentDebts(Budget budget);

    double getNonCurrentDebts(Budget budget);
```

```
18      double getCurrentAssets(Budget budget); //Working capital = current assets - current liabilities

20      double getNonCurrentAssets(Budget budget); // Total Asset = all assets - all liabilities

22      double getInventories(Budget budget); // Quick assest = current assets - inventories

24      double getMarketValueEquity(Budget budget); //only for listed company

26      double getEquity(Budget budget);

28      double getRetainedEarnings(Budget budget);

30      double getCashFlow(Budget budget);

32      double getSales(Budget budget); //gross sales or turnover

34      double getCostOfGoods(Budget budget); //Net sales = Sales - COG

36      double getOperatingCost(Budget budget); //EBIT = Sales - COG - Operating Costs

38      double getInterestCost(Budget budget);

40      double getTax(Budget budget); //Net Profit = EBIT - interest - tax
    }
```

Listing A.7: *Main.java*

```
    package api;
 2  import api.budget.Budget;
    import api.budget.Periodicity;
 4  import api.engine.*;
    import java.io.File;
 6  import java.io.IOException;
    import java.net.InetAddress;
 8  import java.sql.SQLException;
    import java.util.*;
10  import util.FileUtil;
    import util.GeneralUtil;
12  import util.SqlClient;
    import weka.classifiers.Classifier;
14
    public class Main {
16      public static String projectFolder;
        public static String dataFolder;
18      public static String resultFolder;
        public static File packageFile;
20      public static File timeFile;
        public static File contingencyFile;
22      public static File averagePrecisionFile;
        public static List<Engine> engines = new ArrayList<Engine>();
24      public static List<String> tests = new ArrayList<String>();
        public static List<TrainingSet> trainingSets;
26      public static Map<TrainingSet, File> cvFileMap = new HashMap<TrainingSet, File>();
        public static Map<Class, CrossValidation[]> cvMap = new HashMap<Class, CrossValidation[]>();
28      public static Map<Class, File> logFileMap = new HashMap<Class, File>();
        public static String[] companies;
30      public static String[] trainingSetNames;
        private static boolean train = true;
32      private static List<Engine> mostAccurateEngines = new ArrayList<Engine>();
        private static List<Engine> fastestEngines = new ArrayList<Engine>();
34      public static void main(String[] args) throws Exception {
            if (args.length > 0) { // get test Id from arguments
36              for (String arg : args) {
                    try {
38                      tests.add(arg);
                    } catch (Exception e) {
40                  }
                }
```

```
42          } else {
                tests.addAll(Arrays.asList(
44                      "sales - cog - oper_cost - interest - tax", //net income
                        "cash_flow",
46                      "1.2 * work_cap / total_asset + 1.4 * ret_earning / total_asset + 3.3 * ebit / total_asset +
                              0.6 * market_equity / total_liab + 1.0 * sales / total_asset", //z-score
                        "net_income / sales", //ROA
48                      "sales / total_asset", //profit margin
                        "net_income / equity" //ROE
50                      ));
                trainingSetNames = new String[]{"\\testA", "\\testB", "\\testC", "\\testD", "\\testE", "\\testF"};
                     // for Latex only
52          }
            init();
54          //submitBudgets();
            String anzsic = "OO211"; //government
56          Periodicity periodicity = Periodicity.ANNUALLY;
            trainingSets = getTrainingSets(anzsic, periodicity, tests.toArray());
58          engines.addAll(Arrays.asList(new Engine[]{
                        new Weka() {
60                          @Override
                            public Class<? extends Classifier> getClassifierType() {
62                              return weka.classifiers.functions.LibSVM.class;
                            }
64
                        },
66                      new Weka() {
                            @Override
68                          public Class<? extends Classifier> getClassifierType() {
                                return weka.classifiers.functions.SMO.class;
70                          }
                        },
72                      new Weka() {
                            @Override
74                          public Class<? extends Classifier> getClassifierType() {
                                return weka.classifiers.functions.MultilayerPerceptron.class;
76                          }
                        },
78                      new Weka() {
                            @Override
80                          public Class<? extends Classifier> getClassifierType() {
                                return weka.classifiers.functions.RBFNetwork.class;
82                          }
                        },
84                      new Weka() {
                            @Override
86                          public Class<? extends Classifier> getClassifierType() {
                                return weka.classifiers.bayes.NaiveBayes.class;
88                          }
                        },
90                      new Weka() {
                            @Override
92                          public Class<? extends Classifier> getClassifierType() {
                                return weka.classifiers.trees.ADTree.class;
94                          }
                        },
96                      new Weka() {
                            @Override
98                          public Class<? extends Classifier> getClassifierType() {
                                return weka.classifiers.trees.BFTree.class;
100                         }
                        },
102                     new Weka() {
                            @Override
104                         public Class<? extends Classifier> getClassifierType() {
                                return weka.classifiers.lazy.IBk.class;
106                         }
                        },}));
108         String date = "2012-06-30";
            List<InputVector> predictionSet = getInputSet(date, date, Periodicity.ANNUALLY);
110         //prediction results header
```

```
        int nameLength = 34;
        int cellLength = 3;
        String separator = " | ";
        StringBuilder resultsHeader = new StringBuilder();
        resultsHeader.append(GeneralUtil.matchLength("Company", "", nameLength));
        resultsHeader.append(separator).append(GeneralUtil.matchLength("Budget", "", 9));
        for (int j = 0; j < trainingSets.size(); j++) {
            resultsHeader.append(separator).append(
                    GeneralUtil.matchLength("T" + (j + 1),
                    "",
                    cellLength * 2 + separator.length()));
        }
        resultsHeader.append("\n");
        for (int i = 0; i < engines.size(); i++) {
            Engine engine = engines.get(i);
            String engineName = engine.getEngineName();
            String engineId = "E" + (i + 1);
            //for LaTex only
            if (i == 2 || i == 4 || i == 5) {
                mostAccurateEngines.add(engine);
            }
            if (i == 4 || i == 5 || i == 7) {
                fastestEngines.add(engine);
            }
            //
            double E_accuracy = 0;
            double E_precision = 0;
            double E_recall = 0;
            double E_error = 0;
            double E_fmeasure = 0;
            double E_negativePrecision = 0;
            String modelFilePath = dataFolder + engineName;
            String logFilePath = resultFolder + engineId + ".log";
            File logFile = new File(logFilePath);
            FileUtil.emptyFile(logFile);
            logFileMap.put(engine.getClass(), logFile);
            cvMap.put(engine.getClass(), new CrossValidation[trainingSets.size()]);
            engine.log(GeneralUtil.SECTION_BREAK);
            engine.log("Starting Engine: " + engineName + "...");
            //log
            FileUtil.appendToFile(packageFile,
                    "",
                    "",
                    engineId,
                    engineName,
                    engine.getClassifierName());
            //log accuracy by engine
            File accuracyFile = new File(resultFolder + "accuracy" + engineId + ".csv");
            FileUtil.emptyFile(accuracyFile);
            TrainingVector[][] results = new TrainingVector[predictionSet.size()][trainingSets.size()];
            for (int k = 0; k < trainingSets.size(); k++) {
                TrainingSet trainingSet = trainingSets.get(k);
                String testId = "T" + (k + 1);
                //cross validate
                CrossValidation cv = engine.crossValidate(trainingSet);
                cvMap.get(engine.getClass())[k] = cv;
                double accuracy = cv.getAccuracy();
                double precision = cv.getPrecision();
                double recall = cv.getRecall();
                double error = cv.getError();
                double fmeasure = cv.getFMeasure();
                double negativePrecision = cv.getNegativePrecision();
                E_accuracy += accuracy;
                E_precision += precision;
                E_recall += recall;
                E_error += error;
                E_fmeasure += fmeasure;
                E_negativePrecision += negativePrecision;
                Object[] array = new Object[]{
                    GeneralUtil.SIMPLE_FORMATTER.format(accuracy),
                    GeneralUtil.SIMPLE_FORMATTER.format(precision),
```

```
                    GeneralUtil.SIMPLE_FORMATTER.format(recall),
                    GeneralUtil.SIMPLE_FORMATTER.format(error),
                    GeneralUtil.SIMPLE_FORMATTER.format(fmeasure),
                    GeneralUtil.SIMPLE_FORMATTER.format(negativePrecision)
                };
                //log accuracy by engine
                FileUtil.appendToFile(accuracyFile, "", "",
                        GeneralUtil.join(
                        new Object[]{
                            testId,
                            trainingSetNames[k]
                        },
                        array));
                //log accuracy by training set
                FileUtil.appendToFile(cvFileMap.get(trainingSet), "", "",
                        GeneralUtil.join(
                        new Object[]{
                            engine.getEngineName()
                        },
                        array));
                // train model
                if (train) {
                    String modelFile = modelFilePath + "-" + testId + ".model";
                    engine.log(GeneralUtil.SECTION_BREAK);
                    engine.train(trainingSet);
                    // save model
                    engine.saveModel(modelFile);
                    // define the prediction engine
                    Engine predictionEngine = GeneralUtil.newInstance(engine.getClass());
                    // load model from file
                    predictionEngine.loadModel(modelFile);
                    Object[] outputs = predictionEngine.predict(predictionSet);
                    for (int j = 0; j < predictionSet.size(); j++) {
                        TrainingVector trainingVector = new TrainingVector(predictionSet.get(j));
                        trainingVector.setPredictedOutput(outputs[j]);
                        trainingVector.setCrossValidation(cv);
                        results[j][k] = trainingVector;
                    }
                }
            }
            logCvTime(engine);
            logContingency(engine);
            //log prediction results by engine
            File predictionFile = new File(resultFolder + "prediction" + engineId + ".csv");
            FileUtil.emptyFile(predictionFile);
            StringBuilder sb = new StringBuilder();
            for (int j = 0; j < results.length; j++) {
                TrainingVector[] trainingVectors = results[j];
                List predictionLine = new ArrayList();
                for (int k = 0; k < trainingVectors.length; k++) {
                    TrainingVector trainingVector = trainingVectors[k];
                    if (k == 0) {
                        Budget budget = trainingVector.getBudget();
                        String company = budget.getCompany().getName();
                        int yyyy = budget.getEndDate().get(Calendar.YEAR);
                        String year = (yyyy - 1) + "/" + yyyy;
                        sb.append(GeneralUtil.matchLength(company, "...", nameLength));
                        sb.append(separator).append(year);
                        predictionLine.add(company);
                    }
                    Object output = trainingVector.getPredictedOutput();
                    String s = "null";
                    String p = "";
                    if (output != null) {
                        if (output.equals(1.0)) {
                            s = "1";
                            p = GeneralUtil.PERCENT_FORMATTER.format(
                                    trainingVector.getCrossValidation().getPrecision());
                        } else if (output.equals(0.0)) {
                            s = "0";
                            p = GeneralUtil.PERCENT_FORMATTER.format(
```

```
                                trainingVector.getCrossValidation().getNegativePrecision());
                    }
                }
                sb.append(separator).append(GeneralUtil.matchLength(s, "", cellLength));
                sb.append(separator).append(GeneralUtil.matchLength(p, "", cellLength));
                predictionLine.add(s.substring(0, 1) + "(" + p.replace("%", "\\%") + ")");
            }
            sb.append("\n");
            FileUtil.appendToFile(predictionFile, "", "", predictionLine.toArray());
        }
        engine.log(resultsHeader.toString() + sb.toString());
        //log average accuracy by engine
        int K = trainingSets.size();
        String startQuote = ""; //for LaTex
        String endQuote = ""; //for LaTex
        if (mostAccurateEngines.contains(engine)) {
            startQuote = "\\textbf{";
            endQuote = "}";
        }
        FileUtil.appendToFile(averagePrecisionFile, startQuote, endQuote,
                engine.getEngineName(),
                GeneralUtil.SIMPLE_FORMATTER.format(E_accuracy / K),
                GeneralUtil.SIMPLE_FORMATTER.format(E_precision / K),
                GeneralUtil.SIMPLE_FORMATTER.format(E_recall / K),
                GeneralUtil.SIMPLE_FORMATTER.format(E_error / K),
                GeneralUtil.SIMPLE_FORMATTER.format(E_fmeasure / K),
                GeneralUtil.SIMPLE_FORMATTER.format(E_negativePrecision / K));
        // finish
        engine.log("Finished Engine" + engineName);
    }
    closeContingency();
}
private static void init() throws Exception {
    // set the data folder path
    if (InetAddress.getLocalHost().getHostName().toLowerCase().contains("wawa")) {
        projectFolder = "/Users/antonyip/Desktop/Passage/";
    } else {
        projectFolder = "/Users/antonyip/Documents/Academic/Unitec/Master of Computing/Passage/";
    }
    dataFolder = projectFolder + "Data/";
    resultFolder = projectFolder + "Source/Java/Results/";
    packageFile = new File(resultFolder + "package.csv");
    FileUtil.emptyFile(packageFile);
    timeFile = new File(resultFolder + "time.csv");
    FileUtil.emptyFile(timeFile);
    averagePrecisionFile = new File(resultFolder + "average.csv");
    FileUtil.emptyFile(averagePrecisionFile);
    contingencyFile = new File(resultFolder + "contingency.tex");
    FileUtil.emptyFile(contingencyFile);
    initContingency();
}
protected static void submitBudgets() throws Exception {
    SqlClient sqlClient = new SqlClient();
    sqlClient.connect();
    companies = new String[]{
        "Ministry of Education",
        "Department of Corrections",
        "Ministry of Justice",
        "Department of Conservation",
        "Ministry of Foreign Affairs and Trade",
        "Department of Internal Affairs",
        "National Library of New Zealand",
        "Department of Labour",
        "Ministry for Primary Industries",
        "Ministry of Agriculture and Forestry",
        "Archives New Zealand",
        "Statistics New Zealand",
        "Ministry of Health",
        "New Zealand Customs Service",
        "Land Information New Zealand",
        "Ministry of Economic Development",
```

```
            "Ministry of Fisheries",
            "Department of Building and Housing",
            "Parliamentary Service",
            "State Services Commission",
            "Ministry of Science and Innovation",
            "Ministry of Transport",
            "Ministry for Culture and Heritage"
        };
        int[] years = new int[]{2007, 2008, 2009, 2010, 2011, 2012, 2013};
        String[] types = new String[]{"Actual", "Forecast"};
        for (String company : companies) {
            for (int year : years) {
                for (String type : types) {
                    sqlClient.callProcedure("submitNZGovtBudget", company, year, type);
                }
            }
        }
    }
    private static List<TrainingSet> getTrainingSets(String anzsic, Periodicity periodicity, Object... tests)
         throws SQLException, ClassNotFoundException, IOException {
        BudgetCalculator budgetCalculator = new BudgetCalculator();
        List<TrainingSet> sets = new ArrayList<TrainingSet>();
        for (int i = 0; i < tests.length; i++) {
            TrainingSet trainingSet = budgetCalculator.getTrainingSet(
                    null, anzsic, null, null, null, null, null, periodicity, tests[i]);
            trainingSet.setId(i);
            sets.add(trainingSet);
            File cvFile = new File(resultFolder + "cvT" + (i + 1) + ".csv");
            FileUtil.emptyFile(cvFile);
            cvFileMap.put(trainingSet, cvFile);
        }
        return sets;
    }
    private static List<InputVector> getInputSet(String fromDate, String toDate, Periodicity periodicity) throws
          SQLException, ClassNotFoundException {
        BudgetCalculator budgetCalculator = new BudgetCalculator();
        return budgetCalculator.getInputSet(null, fromDate, toDate, periodicity);
    }
    private static void logCvTime(Engine engine) throws IOException {
        CrossValidation[] cvs = cvMap.get(engine.getClass());
        Long totalTime = 0L;
        String startQuote = ""; //for LaTex
        String endQuote = ""; //for LaTex
        if (fastestEngines.contains(engine)) {
            startQuote = "\\textbf{";
            endQuote = "}";
        }
        StringBuilder sb = new StringBuilder();
        sb.append(startQuote).
                append(engine.getEngineName()).
                append(endQuote).
                append(";");
        for (CrossValidation cv : cvs) {
            totalTime += cv.getDuration();
            sb.append(startQuote).
                    append(GeneralUtil.SIMPLE_FORMATTER.format(0.001D * cv.getDuration())).
                    append(endQuote).
                    append(";");
        }
        sb.append(startQuote).
                append(GeneralUtil.SIMPLE_FORMATTER.format(0.001D * totalTime / cvs.length)).
                append(endQuote).
                append("\n");
        FileUtil.appendToFile(Main.timeFile, sb.toString());
    }
    private static void initContingency() throws IOException {
        //contingency table headers
        StringBuilder sb = new StringBuilder();
        sb.append("\\begin{table}[htbp]\n");
        sb.append("\\begin{minipage}{\\textwidth}\n");
        sb.append("\\centering\n");
```

```
        sb.append("\\caption{Contingency Tables}\n");
        sb.append("\\begin{tabular}\n");
        sb.append("{p{3cm}rr");
        for (int i = 0; i < tests.size(); i++) {
            sb.append("|rr");
        }
        sb.append("}");
        sb.append("\n & \\textbf{$TP$} & $FP$");
        for (int i = 0; i < tests.size(); i++) {
            sb.append("&  \\multicolumn{2}{|l}{}");
        }
        sb.append("\\\\");
        sb.append("\n & $FN$ & \\textbf{$TN$}");
        for (int i = 0; i < tests.size(); i++) {
            sb.append("&  \\multicolumn{2}{|c}{T" + (i + 1) + "}");
        }
        sb.append("\\\\");
        FileUtil.appendToFile(contingencyFile, sb.toString());
    }
    private static void logContingency(Engine engine) throws IOException {
        CrossValidation[] cvs = cvMap.get(engine.getClass());
        StringBuilder sb = new StringBuilder();
        sb.append("\\hline \n \\multicolumn{3}{l|}{" + engine.getEngineName() + "}");
        for (CrossValidation cv : cvs) {
            sb.append("& \\textbf{" + cv.getTruePositive() + "} & " + cv.getFalsePositive());
        }
        sb.append("\\\\");
        sb.append("\n \\multicolumn{3}{l|}{}");
        for (CrossValidation cv : cvs) {
            sb.append("& " + cv.getFalseNegative() + " & \\textbf{" + cv.getTrueNegative() + "}");
        }
        sb.append("\\\\");
        FileUtil.appendToFile(Main.contingencyFile, sb.toString());
    }
    private static void closeContingency() throws IOException {
        StringBuilder sb = new StringBuilder();
        sb.append("\\end{tabular}\n");
        sb.append("\\label{tab:contingency}\n");
        sb.append("\\end{minipage}\n");
        sb.append("\\end{table}\n");
        FileUtil.appendToFile(contingencyFile, sb.toString());
    }
}
```

Java Package: api.budget

Listing A.8: *Budget.java*

```
package api.budget;
import api.company.Company;
import java.util.Calendar;
import util.DateUtil;

public class Budget {
    protected Company company;
    protected Periodicity periodicity;
    protected Calendar endDate;
    protected boolean actual;

    public Budget(Company company, Periodicity periodicity, Calendar endDate, boolean actual) {
        this.company = company;
        this.periodicity = periodicity;
        this.endDate = endDate;
        this.actual = actual;
```

```
    }

    public boolean isActual() {
        return actual;
    }

    public Company getCompany() {
        return company;
    }

    public Calendar getEndDate() {
        return endDate;
    }

    public String getEndDateString() {
        return DateUtil.getTimeStamp(endDate, "yyyy-MM-dd");
    }

    public Periodicity getPeriodicity() {
        return periodicity;
    }
}
```

Listing A.9: *ActualBudget.java*

```
package api.budget;
import api.company.Company;
import java.util.Calendar;

public class ActualBudget extends Budget {
    public ActualBudget(Company company, Periodicity periodicity, Calendar endDate) {
        super(company, periodicity, endDate, true);
    }
}
```

Listing A.10: *ForecastBudget.java*

```
package api.budget;
import api.company.Company;
import java.util.Calendar;

public class ForecastBudget extends Budget {
    public ForecastBudget(Company company, Periodicity periodicity, Calendar endDate) {
        super(company, periodicity, endDate, false);
    }
}
```

Listing A.11: *MonthlyActualBudget.java*

```
package api.budget;
import api.company.Company;
import java.util.Calendar;

public class MonthlyActualBudget extends ActualBudget {
    public MonthlyActualBudget(Company company, Calendar endDate) {
        super(company, Periodicity.MONTHLY, endDate);
    }
}
```

Listing A.12: *MonthlyForecastBudget.java*

```
package api.budget;
import api.company.Company;
import java.util.Calendar;

```

```
public class MonthlyForecastBudget extends ForecastBudget {
    public MonthlyForecastBudget(Company company, Calendar endDate) {
        super(company, Periodicity.MONTHLY, endDate);
    }
}
```

Listing A.13: *Periodicity.java*

```
package api.budget;

public enum Periodicity {

    WEEKLY(0.25),

    FORTNIGHTLY(0.5),

    MONTHLY(1),

    QUARTERLY(3),

    ANNUALLY(12);

    public double value;
    private Periodicity(double value) {
        this.value = value;
    }
}
```

Java Package: api.company

Listing A.14: *Company.java*

```
package api.company;

public class Company {
    protected String id;
    protected String anzsic;
    protected String country;
    protected String name;
    protected int employeeSize;

    public Company(String id, String anzsic, String country, String name) {
        this.id = id;
        this.anzsic = anzsic;
        this.country = country;
        this.name = name;
    }

    public Company(String id, String anzsic, String country) {
        this.id = id;
        this.anzsic = anzsic;
        this.country = country;
    }

    public String getAnzsic() {
        return anzsic;
    }

    public String getId() {
        return id;
    }

```

```
    public String getCountry() {
32      return country;
    }
34
    public String getName() {
36      return name;
    }
38  public int getEmployeeSize() {
        return employeeSize;
40  }
    public void setEmployeeSize(int employeeSize) {
42      this.employeeSize = employeeSize;
    }
44 }
```

Listing A.15: *Government.java*

```
  package api.company;
2
  public class Government extends Company{
4
      public Government(String id) {
6         super(id, "00211", "NZL");
      }
8 }
```

Java Package: api.engine

Listing A.16: *CrossValidation.java*

```
  package api.engine;
2 import api.Main;
  import java.io.IOException;
4 import java.lang.reflect.InvocationTargetException;
  import java.util.LinkedList;
6 import java.util.List;
  import java.util.logging.Level;
8 import java.util.logging.Logger;
  import util.DateUtil;
10 import util.FileUtil;
  import util.GeneralUtil;
12
  public class CrossValidation {
14    private Class<? extends Engine> engineType;
      private int truePositive;
16    private int falsePositive;
      private int falseNegative;
18    private int trueNegative;
      private String engineName;
20    private long duration;

22    public CrossValidation(Class<? extends Engine> engineType) {
          this.engineType = engineType;
24        try {
              engineName = GeneralUtil.newInstance(engineType).getEngineName();
26        } catch (Exception ex) {
          }
28    }
      public String getEngineName() {
30        return engineName;
      }
32
```

```
    public synchronized void perform(final TrainingSet trainingSet) throws InstantiationException,
         IllegalAccessException, InterruptedException, NoSuchMethodException, IllegalArgumentException,
         InvocationTargetException {
        String testName = "T" + ( trainingSet.getId() + 1);
        log(GeneralUtil.SECTION_BREAK);
        log("Performing cross validation for " + testName + "...");
        long startTime = DateUtil.getTimeNow();
        for (final TrainingVector testSample : trainingSet) {
            final Engine engine = GeneralUtil.newInstance(engineType);
            new Thread(new Runnable() {
                public void run() {
                    // new training set excluding the test sample
                    TrainingSet newTrainingSet = new TrainingSet();
                    newTrainingSet.setId(trainingSet.getId());
                    for (TrainingVector trainingSample : trainingSet) {
                        if (trainingSample != testSample) {
                            newTrainingSet.add(trainingSample);
                        }
                    }
                    engine.train(newTrainingSet);
                    testSample.setPredictedOutput(engine.predict(testSample));
                    double actual = (Double) testSample.getDesiredOutput();
                    double predicted = (Double) testSample.getPredictedOutput();
                    if (actual >= 0.5) {
                        if (predicted >= 0.5) {
                            addTruePositive();
                        } else {
                            addFalseNegative();
                        }
                    } else {
                        if (predicted >= 0.5) {
                            addFalsePositive();
                        } else {
                            addTrueNegative();
                        }
                    }
                }
            }).start();
        }
        while (getSampleCount() < trainingSet.size()) {
            wait(25);
        }
        List<Object[]> comparision = new LinkedList<Object[]>();
        for (TrainingVector trainingSample : trainingSet) {
            comparision.add(new Object[]{
                        trainingSample.getDesiredOutput(),
                        trainingSample.getPredictedOutput()
                    });
        }
        duration = DateUtil.getTimeNow() - startTime;
        String durationStr = DateUtil.getDurationStr(duration);
        log("...\nCross validation done in " + durationStr + ".");
        log("Cross validation results for " + testName + ":\n"
                + GeneralUtil.toString(new String[]{"Actual", "Predicted"}, comparision, " | ", 9));
        log(GeneralUtil.SECTION_BREAK);
        log("Contingency table for " + testName + ":\n\tTP FP\n\tFN TN\n\t"
                + GeneralUtil.toString(getContingencyTable(), "\n\t", " "));
        log("Overall Accuracy = (TP + TN) / Total = " + getAccuracy());
        log("Precision P = TP / (TP + FP) = " + getPrecision());
        log("Recall R = TP / (TP + FN) = " + getRecall());
        log("Error E = FP / (FP + TN) = " + getError());
        log("F-mearsure F = 2PR / (P + R) = " + getFMeasure());
    }

    public int getFalseNegative() {
        return falseNegative;
    }

    public int getFalsePositive() {
        return falsePositive;
    }
```

```
102
    public int getTrueNegative() {
104         return trueNegative;
    }
106
    public int getTruePositive() {
108         return truePositive;
    }
110
    public Integer[][] getContingencyTable() {
112         return new Integer[][]{
                    new Integer[]{truePositive, falsePositive},
114                 new Integer[]{falseNegative, trueNegative}};
    }
116
    public double getAccuracy() {
118         return (double) (truePositive + trueNegative) / getSampleCount();
    }
120
    public double getPrecision() {
122         return (double) truePositive / (truePositive + falsePositive);
    }
124
    public double getNegativePrecision() {
126         return (double) trueNegative / (falseNegative + trueNegative);
    }
128
    public double getRecall() {
130         return (double) truePositive / (truePositive + falseNegative);
    }
132
    public double getError() {
134         return (double) falsePositive / (falsePositive + trueNegative);
    }
136
    public double getFMeasure() {
138         double P = getPrecision();
            double R = getRecall();
140         return P * R * 2 / (P + R);
    }
142 public long getDuration() {
        return duration;
144 }
    private synchronized void addFalseNegative() {
146         falseNegative++;
    }
148 private synchronized void addFalsePositive() {
        falsePositive++;
150 }
    private synchronized void addTrueNegative() {
152         trueNegative++;
    }
154 private synchronized void addTruePositive() {
        truePositive++;
156 }
    private int getSampleCount() {
158         return truePositive + falseNegative + falsePositive + trueNegative;
    }
160 private void log(Object o) {
        try {
162             FileUtil.appendToFile(Main.logFileMap.get(engineType), "", "", o);
        } catch (IOException ex) {
164             Logger.getLogger(CrossValidation.class.getName()).log(Level.SEVERE, null, ex);
        }
166     System.out.println(o);
    }
168 }
```

Listing A.17: *Engine.java*

```
package api.engine;
import api.Main;
import java.io.IOException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import util.FileUtil;

public abstract class Engine {

    protected static final String DEFAULT_CLASS_COLUMN = "class";

    protected static final String DEFAULT_ATTRIBUTE_PREFIX = "delta_x";
    //
    private CrossValidation crossValidation;

    public void log(Object o) {
        try {
            FileUtil.appendToFile(Main.logFileMap.get(this.getClass()), "", "", o);
        } catch (IOException ex) {
            Logger.getLogger(Engine.class.getName()).log(Level.SEVERE, null, ex);
        }
        System.out.println(o);
    }

    public CrossValidation getCrossValidation() {
        return crossValidation;
    }

    public abstract String getClassifierName();

    public abstract String getEngineName();

    public abstract void loadModel(String file);

    public abstract void saveModel(String file);

    public CrossValidation crossValidate(TrainingSet trainingSet) {
        crossValidation = new CrossValidation(this.getClass());
        try {
            crossValidation.perform(trainingSet);
        } catch (Exception ex) {
            Logger.getLogger(Engine.class.getName()).log(Level.SEVERE, null, ex);
        }
        return crossValidation;
    }

    public abstract Object predict(InputVector input);

    public abstract void train(TrainingSet trainingSet);

    public Object[] predict(List<? extends InputVector> inputs) {
        Object[] outputs = new Object[inputs.size()];
        int i = 0;
        for (InputVector input : inputs) {
            Object output = predict(input);
            outputs[i++] = output;
        }
        return outputs;
    }
}
```

Listing A.18: *InputVector.java*

```
package api.engine;
import api.budget.Budget;

public class InputVector {
    private Object[] input;
```

```
    private Budget budget;

    public InputVector(Object[] input) {
        this.input = input;
    }

    public Object[] getInput() {
        return input;
    }

    public double[] getInputVector() {
        double[] vector = new double[input.length];
        for (int i = 0; i < vector.length; i++) {
            vector[i] = input[i] == null ? 0 : (Double) input[i];
        }
        return vector;
    }

    public Budget getBudget() {
        return budget;
    }

    public void setBudget(Budget budget) {
        this.budget = budget;
    }
}
```

Listing A.19: *TrainingSet.java*

```
package api.engine;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;

public class TrainingSet implements Collection<TrainingVector> {
    private int id;
    private Collection<TrainingVector> list;
    public TrainingSet() {
        this.list = new ArrayList<TrainingVector>();
    }
    public TrainingSet(Collection<TrainingVector> list) {
        this.list = list;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public <T> T[] toArray(T[] a) {
        return list.toArray(a);
    }
    public Object[] toArray() {
        return list.toArray();
    }
    public int size() {
        return list.size();
    }
    public boolean retainAll(Collection<?> c) {
        return list.retainAll(c);
    }
    public boolean removeAll(Collection<?> c) {
        return list.removeAll(c);
    }
    public boolean remove(Object o) {
        return list.remove(o);
    }
    public Iterator<TrainingVector> iterator() {
        return list.iterator();
```

```
    }
    public boolean isEmpty() {
        return list.isEmpty();
    }
    public boolean containsAll(Collection<?> c) {
        return list.containsAll(c);
    }
    public boolean contains(Object o) {
        return list.contains(o);
    }
    public void clear() {
        list.clear();
    }
    public boolean addAll(Collection<? extends TrainingVector> c) {
        return list.addAll(c);
    }
    public boolean add(TrainingVector e) {
        return list.add(e);
    }
}
```

Listing A.20: *TrainingVector.java*

```
package api.engine;
import java.util.Arrays;

public class TrainingVector extends InputVector implements Cloneable {
    private Object desiredOutput;
    private Object predictedOutput;
    private CrossValidation crossValidation;
    public TrainingVector(InputVector inputVector) {
        super(inputVector.getInput());
        setBudget(inputVector.getBudget());
    }

    public TrainingVector(Object[] input, Object desiredOutput) {
        super(input);
        this.desiredOutput = desiredOutput;
    }

    public Object getDesiredOutput() {
        return desiredOutput;
    }

    public Object getPredictedOutput() {
        return predictedOutput;
    }

    public void setPredictedOutput(Object predictedOutput) {
        this.predictedOutput = predictedOutput;
    }

    public double[] getActualVector() {
        double[] inputVector = super.getInputVector();
        double[] vector = Arrays.copyOf(inputVector, inputVector.length + 1);
        vector[inputVector.length] = (Double) getDesiredOutput();
        return vector;
    }

    public double[] getPredictedVector() {
        double[] inputVector = super.getInputVector();
        double[] vector = Arrays.copyOf(inputVector, inputVector.length + 1);
        vector[inputVector.length] = (Double) getPredictedOutput();
        return vector;
    }
    public CrossValidation getCrossValidation() {
        return crossValidation;
    }
    public void setCrossValidation(CrossValidation crossValidation) {
```

```
        this.crossValidation = crossValidation;
    }
}
```

Listing A.21: *Weka.java*

```
package api.engine;
import api.BudgetCalculator;
import java.util.Arrays;
import java.util.logging.Level;
import java.util.logging.Logger;
import weka.attributeSelection.*;
import weka.classifiers.Classifier;
import weka.classifiers.Evaluation;
import weka.core.*;

public abstract class Weka extends Engine {
//    private Class<? extends weka.classifiers.Classifier> classifierType;
    private Classifier classifier;
    private FastVector attributes;
    private FastVector classes;

    public Weka() {
        initClassifier();
        attributes = new FastVector();
        for (String string : BudgetCalculator.getAttributeNames()) {
            attributes.addElement(new Attribute(string));
        }
        classes = new FastVector();
        classes.addElement("0");
        classes.addElement("1");
        attributes.addElement(new Attribute(DEFAULT_CLASS_COLUMN, classes));
    }

    public Classifier getClassifier() {
        return classifier;
    }

    public String[] getOptions() {
        return new String[]{""};
    }

    public abstract Class<? extends Classifier> getClassifierType();

    private void initClassifier() {
        try {
            this.classifier = (Classifier) getClassifierType().newInstance();
            classifier.setOptions(getOptions());
        } catch (Exception ex) {
            Logger.getLogger(Weka.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    @Override
    public void loadModel(String file) {
        try {
            classifier = (Classifier) SerializationHelper.read(file);
        } catch (Exception ex) {
            Logger.getLogger(Weka.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    @Override
    public void saveModel(String file) {
        try {
            SerializationHelper.write(file, classifier);
        } catch (Exception ex) {
            Logger.getLogger(Weka.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    @Override
```

```
64      public Object predict(InputVector input) {
            Instances data = getInstances(input);
66          try {
                return new Evaluation(data).evaluateModelOnce(classifier, data.firstInstance());
68          } catch (Exception ex) {
                Logger.getLogger(Weka.class.getName()).log(Level.SEVERE, null, ex);
70          }
            return null;
72      }
        @Override
74      public void train(TrainingSet trainingSet) {
            Instances data = getInstances(trainingSet);
76  //        setupAttributes(data);
            try {
78              classifier.buildClassifier(data);
            } catch (Exception ex) {
80              Logger.getLogger(Weka.class.getName()).log(Level.SEVERE, null, ex);
            }
82      }
        @Override
84      public String getEngineName() {
            return "Weka 3.6.4-" + classifier.getClass().getSimpleName();
86      }
        @Override
88      public String getClassifierName() {
            return classifier.getClass().getName();
90      }
        private Instances getInstances(TrainingSet trainingSet) {
92          Instances data = new Instances("trainingSet", attributes, 0);
            for (TrainingVector trainingSample : trainingSet) {
94              double[] vars = Arrays.copyOf(trainingSample.getInputVector(), attributes.size());
                int classIndex = attributes.size() - 1;
96              vars[classIndex] = (Double) trainingSample.getDesiredOutput() < 0.5
                        ? classes.indexOf("0")
98                      : classes.indexOf("1");
                data.add(new Instance(1.0, vars));
100         }
            data.setClassIndex(attributes.size() - 1);
102         return data;
        }
104     private Instances getInstances(InputVector input) {
            Instances data = new Instances("inputSet", attributes, 0);
106         double[] vars = Arrays.copyOf(input.getInputVector(), attributes.size());
            int classIndex = attributes.size() - 1;
108         vars[classIndex] = classes.indexOf("0");
            data.add(new Instance(1.0, vars));
110         data.setClassIndex(attributes.size() - 1);
            return data;
112     }
        private void setupAttributes(Instances data) {
114         // setup attribute selection
            AttributeSelection attsel = new AttributeSelection();
116         ASEvaluation eval = new GainRatioAttributeEval();
            ASSearch search = new Ranker();
118         attsel.setEvaluator(eval);
            attsel.setSearch(search);
120         // perform attribute selection
            try {
122             attsel.SelectAttributes(data);
            } catch (Exception ex) {
124             Logger.getLogger(Weka.class.getName()).log(Level.SEVERE, null, ex);
            }
126         int[] indices = new int[0];
            try {
128             indices = attsel.selectedAttributes();
            } catch (Exception ex) {
130             Logger.getLogger(Weka.class.getName()).log(Level.SEVERE, null, ex);
            }
132         System.out.println(
                    "selected attribute indices (starting with 0):\n"
134                 + Utils.arrayToString(indices));
```

```
    }
}
```

Java Package: util

Listing A.22: *DateUtil.java*

```
package util;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.TimeZone;

public class DateUtil {
    public static final long SECOND_TIME = 1000;
    public static final long MINUTE_TIME = 60 * SECOND_TIME;
    public static final long HOUR_TIME = 60 * MINUTE_TIME;
    public static final long DAY_TIME = 24 * HOUR_TIME;
    public static final String DATE_FORMAT = "yyyy-MM-dd";
    public static final String MINUTE_TIME_FORMAT = "HH:mm";
    public static final String SECOND_TIME_FORMAT = MINUTE_TIME_FORMAT + ":ss";
    public static final String DATETIME_FORMAT = DATE_FORMAT + " " + SECOND_TIME_FORMAT;

    public static Calendar getCalendarDate(String dateString, String format) {
        try {
            DateFormat dateFormat = new SimpleDateFormat(format);
            Calendar calendar = Calendar.getInstance();
            long theTime = dateFormat.parse(dateString).getTime();
            calendar.setTimeInMillis(theTime);
            return calendar;
        } catch (ParseException ex) {
            return null;
        }
    }

    public static String getTimeStamp(Calendar date, String format) {
        DateFormat dateFormat = new SimpleDateFormat(format);
        return dateFormat.format(date.getTime());
    }

    public static long getTimeNow() {
        return System.currentTimeMillis();
    }

    public static Calendar getCalendarDate(long milliseconds, TimeZone timeZone) {
        Calendar date = Calendar.getInstance();
        date.setTimeInMillis(milliseconds);
        return date;
    }

    public static Calendar getCalendarDate() {
        return Calendar.getInstance();
    }

    public static String getDurationStr(long milliseconds) {
        return getDurationStr(milliseconds, "day", "hour", "minute", "second");
    }

    public static String getDurationStrSince(long since) {
        return getDurationStr(getTimeNow() - since);
    }
    public static String getSimpleDurationStr(long milliseconds) {
        return getDurationStr(milliseconds, "D", "h", "m", "s");
```

```
58      }

60      public static String getDateString(Calendar calendar) {
            if (calendar == null) {
62              return "";
            }
64          return getTimeStamp(calendar, DATE_FORMAT);
        }
66      public static Calendar clone(Calendar calendar) {
            if (calendar == null) {
68              return null;
            }
70          return (Calendar) calendar.clone();
        }
72
        public static String convertDateString(String dateString, String oldFormat, String NewFormat) throws
             ParseException {
74          return getTimeStamp(getCalendarDate(dateString.replaceAll("\\b\\s{1,}\\b", " "), oldFormat), NewFormat);
        }
76      private static String getDurationStr(long milliseconds, String... units) {
            StringBuilder sb = new StringBuilder();
78          double[] values = {
                (int) milliseconds / DAY_TIME,
80              (int) milliseconds % DAY_TIME / HOUR_TIME,
                (int) milliseconds % HOUR_TIME / MINUTE_TIME,
82              (double) milliseconds % MINUTE_TIME / SECOND_TIME};
            for (int i = 0; i < units.length; i++) {
84              if (i == units.length - 1 || values[i] > 0) {
                    sb.append(sb.length() > 0 ? " " : "").
86                          append(GeneralUtil.SIMPLE_FORMATTER.format(values[i])).
                            append(" ").
88                          append(units[i]).
                            append(values[i] > 1 && units[i].length() > 1 ? "s" : "");
90              }
            }
92          return sb.toString();
        }
94  }
```

Listing A.23: *FileUtil.java*

```
    package util;
2   import java.io.*;
    import java.util.ArrayList;
4
    public class FileUtil {
6
        public static Object[][] readCsvFile(File aFile, String separator) throws IOException {
8           ArrayList<Object[]> array = new ArrayList<Object[]>();
            BufferedReader input = new BufferedReader(new FileReader(aFile));
10          try {
                String line;
12              while ((line = input.readLine()) != null) {
                    String[] row = line.split(separator);
14                  array.add(row);
                }
16          } finally {
                input.close();
18          }
            return array.toArray(new Object[0][0]);
20      }
        public static void saveToFile(File file, String string, boolean append) throws IOException {
22          FileWriter fw = null;
            try {
24              fw = new FileWriter(file, append);
                PrintWriter out = new PrintWriter(fw);
26              out.print(string);
                out.close();
28          } catch (IOException ex) {
```

```
            throw ex;
        } finally {
            try {
                fw.close();
            } catch (IOException ex) {
                throw ex;
            }
        }
    }
    public static void saveToFile(File file, String string) throws IOException {
        saveToFile(file, string, false);
    }
    public static void appendToFile(File file, String string) throws IOException {
        saveToFile(file, string, true);
    }
//    public static void appendToFile(File file, Object... objects) throws IOException {
//        appendToFile(file, GeneralUtil.toString(objects, ";") + "\n");
//    }
    public static void appendToFile(File file, String startQuote, String endQuote, Object... objects) throws
         IOException {
        appendToFile(file, GeneralUtil.toString(objects, startQuote, endQuote, ";") + "\n");
    }
    public static void emptyFile(File file) throws IOException {
        saveToFile(file, "");
    }
}
```

Listing A.24: *GeneralUtil.java*

```
package util;
import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;
import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.util.*;

public class GeneralUtil {

    public static final String LINE_SEPARATOR = System.getProperty("line.separator");

    public static final String SECTION_BREAK = "\n----------------------------------------";

    public static final NumberFormat SIMPLE_FORMATTER = new DecimalFormat("0.###");
    public static final NumberFormat PERCENT_FORMATTER = new DecimalFormat("0%");

    public static <T> T[] join(T[] array1, T[] array2) {
        ArrayList<T> newArray = new ArrayList<T>();
        newArray.addAll(Arrays.asList(array1));
        newArray.addAll(Arrays.asList(array2));
        return newArray.toArray(Arrays.copyOf(array1, 0));
    }

    public static <T> T[] join(T[] array1, T newItem) {
        ArrayList<T> newArray = new ArrayList<T>();
        newArray.addAll(Arrays.asList(array1));
        newArray.add(newItem);
        return newArray.toArray(Arrays.copyOf(array1, 0));
    }

    public static String matchLength(Object object, String symbol, int length) {
        String newString = String.valueOf(object).trim();
        if (length <= 0) {
            return newString;
        }
        if (newString.length() > length) {
            newString = newString.substring(0, length - symbol.length()) + symbol;
        } else {
            while (newString.length() < length) {
                newString += " ";
```

```
            }
        }
        return newString;
    }
    public static String lengthen(Object object, String startQuote, String endQuote, int length) {
        return matchLength(startQuote + String.valueOf(object) + endQuote, "", length);
    }

    public static String toString(Collection collection, String separator, int fixedLength) {
        return toString(collection, "", "", separator, fixedLength);
    }

    public static String toString(Collection collection, String startQuote, String endQuote, String separator,
         int fixedLength) {
        Iterator iter;
        if (collection == null || (!(iter = collection.iterator()).hasNext())) {
            return "";
        }
        StringBuilder sb = new StringBuilder(lengthen(iter.next(), startQuote, endQuote, fixedLength));
        while (iter.hasNext()) {
            sb.append(separator).append(lengthen(iter.next(), startQuote, endQuote, fixedLength));
        }
        return sb.toString();
    }

    public static String toString(Collection collection, String separator) {
        return toString(collection, separator, 0);
    }

    public static String toString(Collection collection) {
        return toString(collection, ", ");
    }

    public static String toString(Object[] objects, String separator, String startQuote, String endQuote, int
         fixedLength) {
        if (objects == null) {
            return "";
        }
        return toString(Arrays.asList(objects), separator, startQuote, endQuote, fixedLength);
    }

    public static String toString(Object[] objects, String separator, int fixedLength) {
        return toString(objects, "", "", separator, fixedLength);
    }

    public static String toString(Object[] objects, String startQuote, String endQuote, String separator) {
        return toString(objects, startQuote, endQuote, separator, 0);
    }

    public static String toString(Object[] objects, String separator) {
        return toString(objects, "", "", separator);
    }

    public static String toString(Object[] objects) {
        return toString(objects, ", ");
    }

    public static String toString(Object[][] objects, String lineSeparator, String colSeparator) {
        if (objects == null) {
            return "";
        }
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < objects.length; i++) {
            if (sb.length() > 0) {
                sb.append(lineSeparator);
            }
            sb.append(toString(objects[i], colSeparator));
        }
        return sb.toString();
    }
```

```
110     public static String toString(Object[][] objects) {
            return toString(objects, LINE_SEPARATOR, ": ");
112     }

114     public static String toString(String[] columnHeads, Object[][] table, String cellSeparator, int cellwidth) {
            return toString(columnHeads, Arrays.asList(table), cellSeparator, cellwidth);
116     }

118     public static String toString(String[] columnHeads, List<Object[]> table, String cellSeparator, int
             cellwidth) {
            StringBuilder sb = new StringBuilder();
120         sb.append(toString(columnHeads, cellSeparator, cellwidth)).append(LINE_SEPARATOR);
            for (Object[] objects : table) {
122             sb.append(toString(objects, cellSeparator, cellwidth)).append(LINE_SEPARATOR);
            }
124         sb.append(table.size()).append(" row");
            if (table.size() > 1) {
126             sb.append("s");
            }
128         return sb.toString();
        }
130     public static void println(Object o) {
            if (o != null) {
132             System.out.println(o.toString());
            }
134     }
        public static <T> T newInstance(Class<T> classType) throws NoSuchMethodException, InstantiationException,
             IllegalAccessException, IllegalArgumentException, InvocationTargetException {
136         Constructor<T> c = classType.getDeclaredConstructor();
            c.setAccessible(true);
138         return c.newInstance();
        }
140 }
```

Listing A.25: *SqlClient.java*

```
    package util;
2   import java.sql.*;

4   public class SqlClient {
        protected String schema = "passage";
6       protected Connection connection;
        public void connect() throws SQLException, ClassNotFoundException {
8           Class.forName("com.mysql.jdbc.Driver");
            connection = DriverManager.getConnection(
10                  "jdbc:mysql://localhost/" + schema,
                    schema,
12                  null);
        }
14      public void disconnect() throws SQLException {
            if (!isConnectionClosed()) {
16              connection.close();
            }
18      }
        public boolean isConnectionClosed() throws SQLException {
20          if (connection == null || connection.isClosed()) {
                return true;
22          } else {
                return false;
24          }
        }
26      public String getSchema() {
            return schema;
28      }
        public ResultSet callProcedure(String procedure) throws SQLException, ClassNotFoundException {
30          return callProcedure(procedure, new Object[]{});
        }
32      public ResultSet callProcedure(String procedure, Object... parameters) throws SQLException,
             ClassNotFoundException {
```

```
        String statement = "CALL `" + procedure + "`("
                + (parameters.length == 0 ? "" : SqlUtil.formatValues(parameters))
                + ");";
        try {
            return connection.createStatement().executeQuery(statement);
        } catch (SQLException e) {
            throw new SQLException(statement, e);
        }
    }
    public boolean execute(String st) throws SQLException, ClassNotFoundException {
        return connection.createStatement().execute(st);
    }
    public int[] executeBatch(String... st) throws SQLException, ClassNotFoundException {
        Statement statement = connection.createStatement();
        for (String s : st) {
            if (!s.isEmpty() && !s.matches("\\s{1,}")) {
                statement.addBatch(s);
            }
        }
        return statement.executeBatch();
    }
    public int executeUpdate(String st) throws SQLException, ClassNotFoundException {
        return connection.createStatement().executeUpdate(st);
    }
    public ResultSet executeQuery(String st) throws SQLException, ClassNotFoundException {
        return connection.createStatement().executeQuery(st);
    }
}
```

Listing A.26: *SqlUtil.java*

```
package util;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.text.ParseException;
import java.util.ArrayList;
import java.util.Calendar;

public class SqlUtil {
    public static String substitute(String script, String[] variables) {
        String newScript = script.trim();
        if (variables != null) {
            for (int i = 0; i < variables.length; i++) {
                newScript = newScript.replaceAll("#" + i + "#", variables[i].replaceAll("'", ""));
            }
        }
        return newScript;
    }
    public static String formatValues(Object... values) {
        if (values == null) {
            return "NULL";
        }
        String dateFormat = DateUtil.DATETIME_FORMAT;
        StringBuilder sb = new StringBuilder();
        for (Object value : values) {
            if (sb.length() > 0) {
                sb.append(", ");
            }
            if (value == null) {
                sb.append("NULL");
            } else if (Boolean.class.isInstance(value)) {
                sb.append((Boolean) value ? "1" : "0");
            } else if (String.class.isInstance(value)) {
                if (((String) value).matches("\\d{8}\\s{1,}\\d{2}:\\d{2}:\\d{2}")) {
                    try {
                        sb.append("'").append(DateUtil.convertDateString((String) value, "yyyyMMdd HH:mm:ss",
                             dateFormat)).append("'");
                    } catch (ParseException ex) {
                        sb.append("'").append(value).append("'");
```

```
38                  }
                } else if (((String) value).equalsIgnoreCase("null")) {
40                  sb.append("NULL");
                } else {
42                  sb.append("'").append(value).append("'");
                }
44          } else if (Calendar.class.isInstance(value)) {
                sb.append("'").append(DateUtil.getTimeStamp((Calendar) value, dateFormat)).append("'");
46          } else if (Long.class.isInstance(value)) {
                sb.append((Long) value == Long.MAX_VALUE ? "NULL" : value);
48          } else if (Integer.class.isInstance(value)) {
                sb.append((Integer) value == Integer.MAX_VALUE ? "NULL" : value);
50          } else if (Number.class.isInstance(value)) {
                sb.append((Double) value == Double.MAX_VALUE
52                      || Double.isNaN((Double) value)
                        ? "NULL" : value);
54          } else {
                sb.append("'").append(value).append("'");
56          }
        }
58      return sb.toString();
    }
60  public static Object[][] getTable(ResultSet resultSet, String... columnNames) throws SQLException {
        ArrayList<Object[]> table = new ArrayList();
62      while (resultSet.next()) {
            Object[] row = new Object[columnNames.length];
64          for (int i = 0; i < columnNames.length; i++) {
                row[i] = resultSet.getObject(columnNames[i]);
66          }
            table.add(row);
68      }
        resultSet.beforeFirst();
70      return table.toArray(new Object[0][0]);
    }
72  public static Object[] quote(Object[] data, String quote) {
        Object[] newData = new Object[data.length];
74      for (int i = 0; i < data.length; i++) {
            Object obj = data[i];
76          if (obj instanceof String) {
                newData[i] = quote((String) obj, quote);
78          } else if (obj instanceof Double) {
                if (!Double.isNaN((Double) obj)) {
80                  newData[i] = obj;
                }
82          } else {
                newData[i] = obj;
84          }
        }
86      return newData;
    }
88  public static String quote(String str, String quote) {
        if (str.matches(quote + ".+" + quote)) {
90          return str;
        } else {
92          return quote + str + quote;
        }
94  }
}
```

Appendix B

Experiment Results

Table B.1: *Cross Validation Summary for Net Profit Prediction*

Engine	A	P	R	E	F	NP
Weka 3.6.4-LibSVM	0.8	0.8	1	1	0.889	
Weka 3.6.4-SMO	0.783	0.797	0.979	1	0.879	0
Weka 3.6.4-MultilayerPerceptron	0.817	0.863	0.917	0.583	0.889	0.556
Weka 3.6.4-RBFNetwork	0.683	0.774	0.854	1	0.812	0
Weka 3.6.4-NaiveBayes	0.717	0.897	0.729	0.333	0.805	0.381
Weka 3.6.4-ADTree	0.717	0.804	0.854	0.833	0.828	0.222
Weka 3.6.4-BFTree	0.733	0.796	0.896	0.917	0.843	0.167
Weka 3.6.4-IBk	0.717	0.83	0.812	0.667	0.821	0.308

Table B.2: *Cross Validation Summary for Cash Flow Prediction*

Engine	A	P	R	E	F	NP
Weka 3.6.4-LibSVM	0.583	0.583	1	1	0.737	
Weka 3.6.4-SMO	0.583	0.583	1	1	0.737	
Weka 3.6.4-MultilayerPerceptron	0.5	0.581	0.514	0.52	0.545	0.414
Weka 3.6.4-RBFNetwork	0.517	0.571	0.686	0.72	0.623	0.389
Weka 3.6.4-NaiveBayes	0.433	0.516	0.457	0.6	0.485	0.345
Weka 3.6.4-ADTree	0.633	0.697	0.657	0.4	0.676	0.556
Weka 3.6.4-BFTree	0.55	0.577	0.857	0.88	0.69	0.375
Weka 3.6.4-IBk	0.533	0.595	0.629	0.6	0.611	0.435

Table B.3: *Cross Validation Summary for z-score Prediction*

Engine	A	P	R	E	F	NP
Weka 3.6.4-LibSVM	0.667	0.642	0.971	0.76	0.773	0.857
Weka 3.6.4-SMO	0.7	0.667	0.971	0.68	0.791	0.889
Weka 3.6.4-MultilayerPerceptron	0.667	0.692	0.771	0.48	0.73	0.619
Weka 3.6.4-RBFNetwork	0.717	0.714	0.857	0.48	0.779	0.722
Weka 3.6.4-NaiveBayes	0.717	0.696	0.914	0.56	0.79	0.786
Weka 3.6.4-ADTree	0.7	0.73	0.771	0.4	0.75	0.652
Weka 3.6.4-BFTree	0.767	0.756	0.886	0.4	0.816	0.789
Weka 3.6.4-IBk	0.7	0.707	0.829	0.48	0.763	0.684

Table B.4: *Cross Validation Summary for Profit Margin Prediction*

Engine	A	P	R	E	F	NP
Weka 3.6.4-LibSVM	0.817	0.817	1	1	0.899	
Weka 3.6.4-SMO	0.8	0.814	0.98	1	0.889	0
Weka 3.6.4-MultilayerPerceptron	0.833	0.882	0.918	0.545	0.9	0.556
Weka 3.6.4-RBFNetwork	0.833	0.842	0.98	0.818	0.906	0.667
Weka 3.6.4-NaiveBayes	0.733	0.923	0.735	0.273	0.818	0.381
Weka 3.6.4-ADTree	0.767	0.857	0.857	0.636	0.857	0.364
Weka 3.6.4-BFTree	0.85	0.857	0.98	0.727	0.914	0.75
Weka 3.6.4-IBk	0.767	0.857	0.857	0.636	0.857	0.364

Table B.5: *Cross Validation Summary for DuPont ROA Prediction*

Engine	A	P	R	E	F	NP
Weka 3.6.4-LibSVM	0.683		0	0		0.683
Weka 3.6.4-SMO	0.683		0	0		0.683
Weka 3.6.4-MultilayerPerceptron	0.65	0.417	0.263	0.171	0.323	0.708
Weka 3.6.4-RBFNetwork	0.7	0.545	0.316	0.122	0.4	0.735
Weka 3.6.4-NaiveBayes	0.733	0.636	0.368	0.098	0.467	0.755
Weka 3.6.4-ADTree	0.7	0.533	0.421	0.171	0.471	0.756
Weka 3.6.4-BFTree	0.667	0.444	0.211	0.122	0.286	0.706
Weka 3.6.4-IBk	0.6	0.368	0.368	0.293	0.368	0.707

Table B.6: *Cross Validation Summary for Return on Equity Prediction*

Engine	A	P	R	E	F	NP
Weka 3.6.4-LibSVM	0.8	0.8	1	1	0.889	
Weka 3.6.4-SMO	0.783	0.797	0.979	1	0.879	0
Weka 3.6.4-MultilayerPerceptron	0.817	0.863	0.917	0.583	0.889	0.556
Weka 3.6.4-RBFNetwork	0.683	0.774	0.854	1	0.812	0
Weka 3.6.4-NaiveBayes	0.717	0.897	0.729	0.333	0.805	0.381
Weka 3.6.4-ADTree	0.717	0.804	0.854	0.833	0.828	0.222
Weka 3.6.4-BFTree	0.717	0.792	0.875	0.917	0.832	0.143
Weka 3.6.4-IBk	0.717	0.83	0.812	0.667	0.821	0.308