Classification System for Mortgage Arrear Management

Zhe Sun

S2028091

December 2013

Master Project

Computing Science, University of Groningen, the Netherlands

Internal supervisors:

Prof. Nicolai Petkov (Computing Science, University of Groningen)
Dr. Marco Wiering (Artificial Intelligence, University of Groningen)

External supervisor:

Wendell Kuiling (Business Change, ING Domestic Bank)
Jaap Meester (Business Change, ING Domestic Bank)
Jurriaan Tressel (Business Intelligence & Analytics, Deloitte)

university of groningen / faculty of mathematics and natural sciences

Contents

Abstract

Background The ING Domestic Bank possesses around 22% market share of Dutch mortgages. Normally, mortgage customers have to pay the interest or deposit monthly. But somehow, a considerable number of customers repay late, or default for one or even several months, which brings tremendous losses to ING. The Arrears department manages the arrears of mortgage payments, and it contacts defaulters by letters, SMS, Emails or phone calls. Comparing with the existing working process, the Arrears department aims at to make the treatments more intensive in order to push defaulters to repay as soon as possible, while keeping the current operational cost.

Research problem We develop a classification model to predict the behaviour of the mortgage customers who were healthy in the last month but do not pay the debt at the beginning of the current month. One label with two possible values is assigned by our model: the delayers, who just pay late but not exceeding 1 month, and defaulters, who do not pay even at the end of the month. In this way, the Arrears department can only treat defaulters intensively, who really have payment problems.

Data and method In this project, 400,000 customers with more than 2,000 features are collected from the ING data warehouse. Feature selection and data preprocessing are executed first. Then, we train several popular basic classifiers such as KNN, Naive Bayes, decision trees, logistic regression, and also some ensemble methods like bagging, random forests, boosting, voting and stacking. Since the two classes are highly imbalanced (the ratio of defaulters to delayers is around 1:9), we discuss the evaluation metrics of skewed data learning. The Area under the ROC curve is employed to compare the results of different classifiers. Besides, the impacts of sampling techniques are empirically studied as well.

Result and conclusion Our experiments show that ensemble methods increase the performance of basic classifiers remarkably. We also conclude that symmetric sampling improves the classification performance. Balanced random forests is chosen to build the model for the Arrears department, which gives an AUC value of around 0.772. The model has already been deployed into the daily work of the Arrears department of the ING domestic bank since June 2013. Finally, cost matrix analysis and feature importance ranking are studied in order to guide the daily work of the Arrears department and give a deep insight to this problem. Conservatively estimating, the risk cost of [redacted] can be saved per month by using the model and the new working process.

Chapter 1

Introduction

1.1 Current situation and complication of the Arrears department of ING

The ING Group (Dutch: ING Groep) is a global financial institution of Dutch origin offering banking, investments, life insurance and retirement services [1]. The ING Domestic Bank Netherlands is the retail bank of the ING Group in the Netherlands, which is subsidiary of the ING Group. Where in this thesis ING is used, ING Domestic Bank Netherlands is meant.

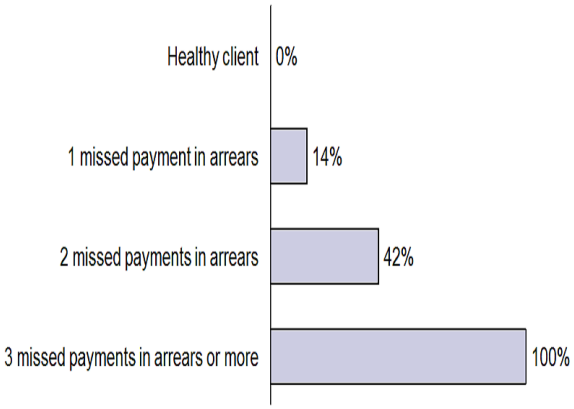
ING owns more than 700,000 mortgages in the Netherlands, which is a share of 22% in the Dutch mortgages market [2]. Normally, customers have to pay the interest or deposit of mortgages monthly. There are four opportunities each month that ING deducts the payment from each customer's appointed bank account (one chance of collecting money is called one "incasso" in Dutch). Most customers pay on the first "incasso" on schedule, but somehow, around [redacted] customers (In consideration of the total number of mortgages, [redacted] is a considerable amount) pay late, or even default for one or several months. Figure 1.1 illustrates four typical payment behaviors of customers who are supposed to pay €200 per month.

	March,2013					April,2013					May,2013				
	5th	12th	19th	25th	Arrears	6th	13th	20th	26th	Arrears	5th	12th	19th	25th	Arrears
Customer A	€200				**€0**	€200				**€0**	€200				**€0**
Customer B	€0	€200			**€0**	€0	€0	€0	€200	**€0**	€0	€0	€200		**€0**
Customer C	€0	€0	€0	€0	**€200**	€0	€400	€0	€0	**€0**	€0	€0	€0	€0	**€200**
Customer D	€0	€0	€0	€0	**€200**	€0	€0	€0	€0	**€400**	€0	€0	€0	€0	**€600**

Figure 1.1: Customer A always pays in the first "incasso" of each month; customer B could not catch up the first "incasso" sometimes, but does not have arrears; customer C defaults in March, repays all debts and arrears in April, but defaults in May again; customer D could not repay for all three months and has €600 arrears in the end of May.

Due to the bad economic situation in the recent years, more and more customers meet financial distress and stay in arrears. The EU and the Dutch government have already strengthened the supervision on the mortgage sector, and the capital and liquidity requirements for banks under Basel III have become stricter [3]. One policy is loan loss provision illustrated in figure 1.2a. In the Netherlands, if one mortgage customer misses 1, 2 or more than 3 monthly payments, the bank has to freeze 14%, 42% and 100% of the total value of the mortgage as guarantee, respectively.

Figure 1.2b compares the average recovery rate of customers in arrears between ING (the orange curve) and the competitors in the Dutch mortgages market: the leftmost point of the curves means the whole population who have arrears in the first month; customers repay and go out of arrears gradually, so curves fall down month by month; eventually some customers still stick to the arrears even after a whole year. It is clear that ING customers have a poor recovery rate and this brings ING tremendous costs.



(a) Provision ("Mutatie in voorzieningen" in Dutch); a certain amount of money is frozen and used as provisions.

(b) Recovery rates of ING customers and benchmark

Figure 1.2

In ING, the Arrears Department is in charge of managing the arrears of mortgage payments. It starts tracing customers when they have arrears (miss the first "incasso" or have been in arrears before). Letters, emails, or SMS will be sent in order to remind customers to pay when they miss the first "incasso"; if customers do not pay longer than one and half months, case managers of the Arrears Department will contact them by phone calls, and help them clear the defaulted payments. As soon as customers repay all defaulted payments, they will be called healthy or out of arrears, and the Arrears Department will not contact them any more.

Besides the aforementioned loan loss provision, the customers who have arrears also cause interest losses and potential collection losses for ING (the details are in appendix A). These tremendous losses compel the Arrears Department to optimize its working flow, e.g., making the orange curve in figure 1.2b go downwards. Intuitively, a stricter treatment will definitely push customers out of arrears. For instance, case managers can start calling customers as soon as they miss the first "incasso": the "lazy" customers who forget to pay will get a reminder; the "unwilling" customers who want to divert money to vacations or fancy electricity appliances will control their consumption desire better; the "bankrupt" customers who indeed meet financial distress will be identified at the very beginning so that ING can offer them a more rational payment schedule. However, contacting every customer in arrears is a very unrealistic task. Neither ING has such capacity of case managers, nor it is willing to pay extraordinary extra money to hire more case managers. So, a smarter process should be adopted. The goal is maintaining the operational cost at the same level while lowering the risk cost, namely pushing customers out of arrears while keeping the same capacity of case managers.

1.2 Research questions

In order to clarify the research questions easily, three definitions are raised as below:

- **New arrear customer**: the customers who were healthy in the previous month and do not pay in the first "incasso" of the current month.

- **Defaulters**: new arrear customers who do **not** repay even at the end of the current month (miss all four "incasso's").
- **Delayed customers**: new arrear customers who repay before the end of the current month (repay in second, third or fourth "incasso").

New arrear customers consist of defaulters and delayers. Let us give examples by the customers in figure 1.1. Suppose May 2013 is the current month: B and C are new arrear customers; B is a delayer, because he/she had no arrears in April, does not repay in the first "incasso" but repays in the third "incasso"; C is a defaulter, because he/she had no arrears in April, and does not repay in whole May.

Currently, the Arrears Department wants to improve the process only for new arrear customers. Customers who are healthy (A in figure 1.1) are out of scope, because they do not bring any risk cost to ING. Customers who have been in arrears for more than one month (D in figure 1.1) are not the central issue either, because these customers have already been traced and contacted by case managers.

On average, ING has around [redacted] new arrear customers in arrears per month, of which nearly 10% are defaulters and 90% are delayers. Intuitively, a defaulter needs extra help from case managers, because delayers repay spontaneously before the end of the month. So, if there is a model which can label new arrear customers as defaulters and delayers at the beginning of each month, they can be treated discriminately: automatic treatment like letter, Email, SMS are sent to delayers, and case managers will give intensive contact or a fine to defaulters.

By implementing the above proposed model two research questions are attempted to be answered:

1. **Can we build a binary classification model to classify the new arrear customers as defaulters and delayers?**
2. **If the answer to question 1 is positive, can the model also have good interpretability so that some rules, patterns or useful knowledge can be used by the Arrears department?**

1.3 Outline

The outline of this thesis is as follows.

Chapter 2 will introduce the framework of our classification system. First, we will review the general pipeline. Then, some details will be described, such as the scope and the availability of the data, data gathering and data preprocessing in this project. Next, a literature review will look back at some approaches in the banking mortgage default field. Last, four popular classifiers, namely case-based reasoning, logistic regression, naive Bayes and decision trees will be described.

Chapter 3 will focus on imbalanced learning. The imbalance of our data is first discussed. Then, various assessment metrics are compared and selected as our evaluation metrics. Next, some sampling techniques are introduced. At last, groups of experiments are set up to compare the results with and without sampling techniques.

Chapter 4 will continue discussing imbalanced learning, but focuses on the effect of ensemble methods. First, some typical ensemble methods like bagging, random forests and boosting will be reviewed and compared with basic classifiers. Then, we will experimentally study the combination of sampling techniques and ensemble methods. Finally, the voting and stacking methods will be researched to examine if this can lead to even better results.

Chapter 5 will first decide on the best classifier according to the test results of the previous chapters. Then, some knowledge will be discovered and interpreted from some aspects such as cost matrix analysis, feature importance, dummy variable analysis and the rules derived from decision trees.

Chapter 6 will conclude the thesis and answer the research questions posed in the first chapter. It will also discuss what ends were left open and give suggestions for further research into model developing and deploying.

Chapter 2

The data

2.1 Classification system framework

Data mining, also known as "knowledge discovery in a database", is the process of discovering interesting patterns in a database that are useful in decision making. Today, with greater data storage capabilities and declining costs, data mining has offered organizations a new way of doing business. Data mining can help organizations better understand their business, be able to better serve their customers, and increase the effectiveness of the organization [4]. According to the investigation of Liao et al. in [5], in the past few decades many organizations in the finance and banking field have recognized the importance of the information they have from their customers. Hormozi and Giles [6] list some typical applications in the banking and retail industries, such as marketing, risk management, fraud detection, customer acquisition and retention.

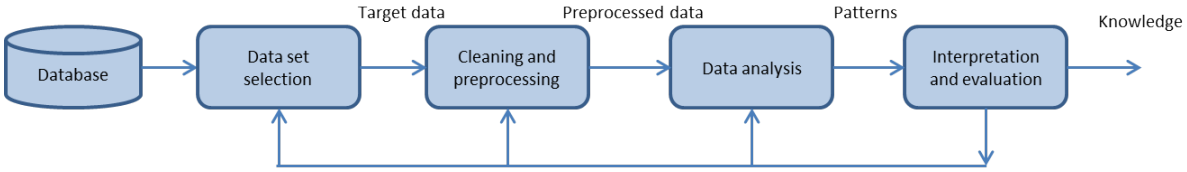


Figure 2.1: An overview of the data mining framework.

Figure 2.1 illustrates a typical data mining workflow. The first step is data selection and gathering for analysis. The data set may be retrieved from a single source, such as a data warehouse, or may be extracted from several operational databases. Then, data cleaning is a prerequisite, because discrepancies, inconsistencies and missing data always exist in real banking databases due to imperfect client information or unsatisfactory database design. Preprocessing techniques such as discretization, normalization, scaling and dimensionality reduction are also required by the following steps. Next, the data set is analysed to generate models that can predict or classify. The model is then validated with new data sets to ensure its generalizability. A bunch of models will be developed, such as statistical approaches and machine learning approaches, to identify patterns in data. At last, some models can be translated into rules or meaningful business knowledge so that they can be comprehended and applied into business processes.

2.2 Data collection and preparation

In this section, how to implement data gathering, initial selection, feature extraction, and data processing will be discussed.

2.2.1 Data review

[redacted]

Like all classification system exercises, the identification of relevant data in the right quantity is critical for the development of meaningful models. Given this and after discussing with the domain experts, we proceeded to identify the necessary data sources available and those readily accessible for initial review. Table 2.1 summarizes the data sources identified and their descriptions.

Data name	Data source	Description
[redacted]	[redacted]	[redacted]

Table 2.1: Initial selection of relevant data from all possible data sources.

Besides the static data listed in table 2.1, some synthetic features are also helpful to our system:

[1] How is 30% computed: we compare the percentage of new arrear customers who have only two or less ING products.

- [redacted]
- [redacted]

Tracing back to a certain number of months from the current month and sampling the static features, a bunch of time series can be obtained. Time series are instructive to the system as well, e.g., the trend of incomes, month-end balance and utility bill in the last 12 months are very good indicators when the financial circumstance of the customer fluctuates.

2.2.2 Data period identification

One rule of thumb in machine learning is that more data beats a more clever algorithm [7]. Given the data availability and its time periodicity, we should collect as many data instances as possible. The Arrears department started tracking repayments of mortgage customers [redacted][3], so the earliest new arrear customers were in [redacted] However, since history data and time series need to be extracted as mentioned in the previous section, the first available month is [redacted]. Till [redacted], new arrear customers which have ever been in the database are available as shown in figure 2.2.

[redacted]

Figure 2.2: Time span of new arrear customers

2.2.3 Data gathering

The aim of data gathering is joining dispersed features from various kinds of sources in table 2.1 and assembling them into one data table. The procedure is illustrated in figure 2.3.

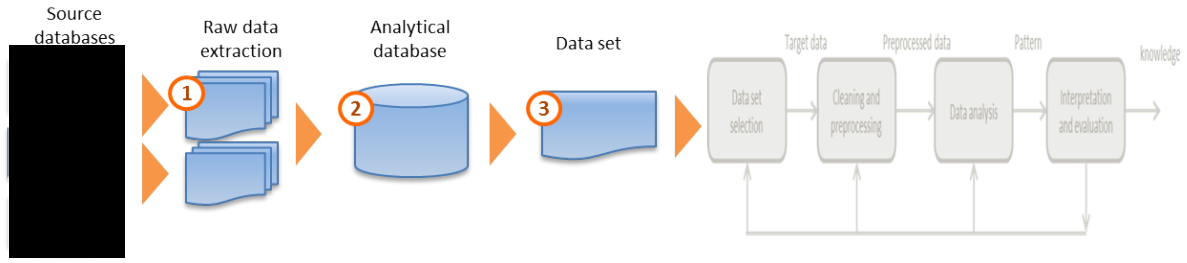


Figure 2.3: Typical analytical architecture. Step 1: Extract required data from the source databases by using SQL queries. Step 2: various data are merged into the analytical database and organized by time and customer ID; history data and time series data are also generated. Step 3: According to different output requirements, data tables are exported to plain text files.

An automatic database platform was built to fulfill data gathering. By running a set of SQL scripts with modifications of a few parameters, the datasets with desired time periods and features are exported from it. In this way, rapid development and monthly operation become reality. Figure 2.4 illustrates the design of the automatic database platform.

The same as any other large scale data warehouse, [redacted] is not organized as a unique large flat table but as a star pattern, where each table contains data that come from different systems. Different

[2]The definition of default term is total arrears divided by the amount of monthly payments. So, term is different from number of defaulted months. It could be a decimal number.

[3]To be precise, the Arrears department employed another way of tracking mortgage customers before [redacted]

tables are linked to each other by unique keys. [redacted] When predicting each month in actual operations, the test set can be obtained automatically as well.


[redacted]

Figure 2.4: The framework of data translation.

It should be emphasized that the variable data in the dataset are always 2 months earlier than the sampling period. For example, regarding new arrear customers in March 2013, features such as salary, month-end balance and transaction records are actually the data of January 2013. This is because the ORION data warehouse spends one month to integrate data from various systems. This limitation causes that variables cannot reflect real-time personal financial information, which deteriorates the quality of the data.

2.3 Feature selection

The initial dataset contains around 2,000 features. It is obviously impossible to use all of them from the perspective of either theory (the curse of dimensions) or practise (CPU and memory resources). This section will discuss dimensionality reduction.

Domain knowledge Before selecting features by using a machine learning approach, we should first ask ourselves "do we have domain knowledge?" If yes, construct a good set of "ad hoc" features [8].

Table 2.2 shows some empirical reasons that customers stay in arrears, which are from the investigation of case managers in the Arrears Department. The corresponding features on the right column in the table will be employed in the system regardless of the result of feature selection.

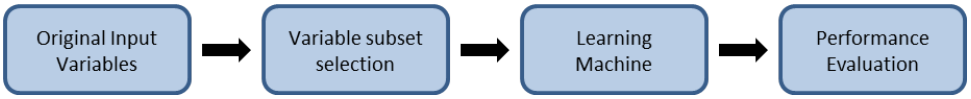
Reasons	Features
[redacted]	[redacted]

Table 2.2: Domain knowledge on the reasons of default and corresponding features.

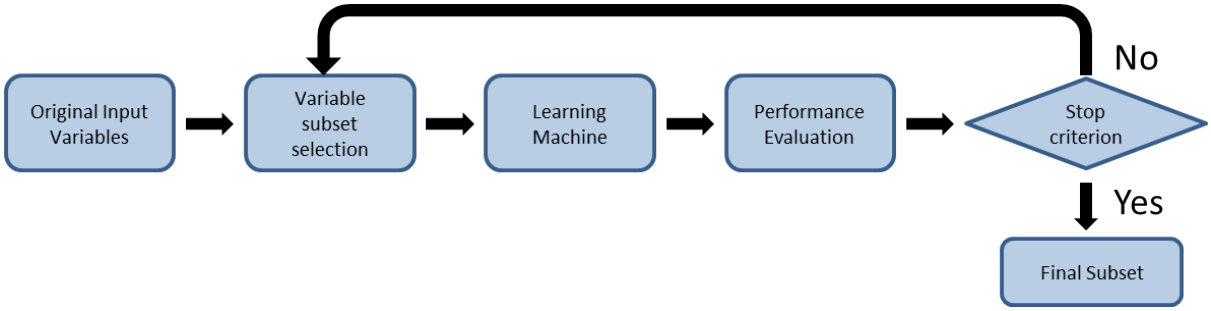
Filter and wrapper methods In literature feature selection methods are classified into three categories: filter, wrapper and embedded methods. The following paragraphs only discuss filter and wrapper methods, and embedded method will be described in chapter 5.

The filter method is a preprocessing phase which is independent of the learning algorithm that is adopted to tune and/or build the model. As shown in figure 2.5a, all input variables are classified on the basis of their pertinence to the target considering statistical tests [10]. The filter method does not involve any training or learning algorithm, so it is computationally convenient especially for large scale data sets. On the other hand the main disadvantage of a filter approach is that, being independent of the algorithm that is used to tune or build the model which is fed with the selected variables as inputs, this method cannot optimize the adopted model in the system [11]. Common feature ranking techniques are information gain, gini-index, relief, χ^2, correlation criterion, etc.

Kohavi and John popularized the wrapper method in [12] in 1997. It considers the learning algorithms as a black box in order to select subsets of variables on the basis of their predictive power. Figure 2.5b illustrates a generic scheme. At first, a subset is generated based on the chosen starting point and searching strategy, for example, best-first, branch-and-bound, simulated annealing, genetic algorithm [12]. Then, a predictor or classifier is employed to evaluate the performance. How to assess the performance needs to be defined as well, e.g., classification accuracy or area under ROC curve. If the performance meets the stopping criterion such as the improvement of classification accuracy less than



(a) Filter



(b) Wrapper

Figure 2.5: Diagram of filter and wrapper approaches [9].

0.01%, the procedure stops; otherwise a new subset of features is generated by the searching algorithm and input to the learner. Compared to filter methods, a wrapper method is simple and universal. On the other hand, a wrapper method is computationally expensive, especially with a large number of features.

Weighted rank voting and first selection Based on the description of filter and wrapper method, we can see that it is wise to use a filter method to do a first selection owing to the large scale data (2000 initial features and more than 400,000 instances). As aforementioned, there are several popular filter approaches such as information gain, Gini-index, relief, χ^2, correlation criterion. Stemming from the field of ensemble learning, some ensemble feature selection techniques were proposed by Saeys et al. in [13] and Shen et al. in [14]. Waad et al. investigated majority vote and mean aggregation of filter approaches in [15], and indicated that there is a general beneficial effect of aggregating feature ranking in credit scoring applications.

In this thesis, we adopt the weighted voting approach in [13]: Consider an ensemble E consisting of s feature selectors, $E = \{F_1, F_2, \cdots, F_s\}$, then we assume each F_i provides a feature ranking $\boldsymbol{f}_i = (f_i^1, \cdots, f_i^N)$, which are aggregated into a consensus feature ranking $\boldsymbol{f}$ by equal weighted voting:

$$\boldsymbol{f}^l = \sum_{i=1}^{s} w(\boldsymbol{f}_i^l)$$

where $w(\cdot)$ denotes a weighting function. In the first selection step, we choose information gain, Gini-index and χ^2 as basic rankers and use equal weights. As a result, 100 features come to the fore from all initial features.

2.4 Time series representation

As mentioned in section 2.2.1, 18 time series which mainly cover the financial situation of customers are sampled. There are several popular time series feature extraction methods such as Discrete Fourier Transform (DFT), Discrete Wavelet Transform (DWT) [16], Symbolic Aggregate approXimation (SAX) [17], Piecewise Aggregate Approximation (PAA) [18]. The common idea behind these methods is that processed time series can be analyzed by classifiers easily, i.e., dimensionality reduction, convenient distance computation and better representation of time series.

In this thesis, we employ a light-weight method to represent time series as one nominal feature. We notate the time series C of length 12 (12 month samples) by a vector $c_1, \cdots, c_{12}$. The method is demonstrated below:

1. Smooth the time series by a n-month width sliding window: $\overline{c_i} = \frac{1}{n}\sum_{j=i}^{i+n-1} c_j$ (we use n = 2 in the system).
2. Normalise the time series to have a mean of zero and standard deviation of one.
3. Given the normalized time series that have a Gaussian distribution, we can simply determine the "breakpoints" that will produce α equal-sized areas under the Gaussian curve [17]. Table 2.3 gives the breakpoints for values of α from 3 to 10.
4. Discretize the last element of the smoothed time series vector $\overline{C}$ according to the value of α and "breakpoints".

β_i \ α	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
β_1	-0.43	-0.67	-0.84	-0.97	-1.07	-1.15	-1.22	-1.28
β_2	0.43	0	-0.25	-0.43	-0.57	-0.67	-0.76	-0.84
β_3		0.67	0.25	0	-0.18	-0.32	-0.43	-0.52
β_4			0.84	0.43	0.18	0	-0.14	-0.25
β_5				0.97	0.57	0.32	0.14	0
β_6					1.07	0.67	0.43	0.25
β_7						1.15	0.76	0.52
β_8							1.22	0.84
β_9								1.28

Table 2.3: A lookup table that contains the breakpoints that divide a Gaussian distribution in an arbitrary number from (3 to 10) of equiprobable regions [17]. Definition of *breakpoints*: breakpoints are a sorted list of numbers $B = \beta_1, \cdots, \beta_{\alpha-1}$ such that the area under a $N(0,1)$ Gaussian curve from β_i to $\beta_{i+1} = 1/\alpha$ (β_0 and β_α are defined as $-\inf$ and $\inf$, respectively).

This extracted feature indicates the latest trends of the customer of salary, balance, utility payment series or other time series on which the feature extraction method applies. For instance, if using 10 level categories ($\alpha = 10$), a value with category 10 or 0 manifests abnormal lift or drop. Domain knowledge points out that the change of financial related context of customers leads to default.

One essential point of learning time series in classifiers is distance measurement: independent of using Euclidean distance, non-linear alignment distance measurement or dynamic time warping, a distance function *dist*() is required to calculate the similarity of time series. In contrast, the extracted feature is more flexible to be adopted in the existing classifiers. Without any extra step, the extracted feature can be embedded into the dataset with other features in the role of either nominal or numeric feature.

2.5 Data preprocessing

So far, 100 selected features, which include around 20 domain knowledge features and 18 extracted features from time series, are ready to be fed into the preprocessing step of the classification system. Table 2.4 lists some of them.

The preprocessing step is indispensable to resolve several types of problems including noisy data, redundant data, missing data values, etc. All the next learning algorithms rely heavily on the product of this stage, which is the final training set. It is noteworthy that different preprocessing methods will be used with different classifiers, for example, discretization is a compulsory step for Naive Bayes; normalization needs to be done for some distance-based or metrics sensitive classifier like kNN, linear regression, neural networks; dummy variable can be employed by logistic regression. Data cleaning, missing value imputation, discretization, normalization will be covered in the next paragraphs and during the discussion of classifiers in section 3.

Data cleaning Data cleaning, also known as instance selection, is the process of removing unwanted instances from a database. Similar to feature selection, data cleaning approaches fall into two categories, filter and wrapper. Wrapper approaches explicitly evaluate results by using the specific machine learning

Features	Description	Features	Description
[redacted]			

Table 2.4: Descriptions of the selected features

algorithm to trigger instance selection, e.g., Tomek-Link sample method, edited nearest neighbor (ENN) rule, etc. Chandola et al. provided a comprehensive overview of existing techniques in [19]. The filter approach is more straightforward: suspicious instances will be evaluated and removed instance by instance. Several filter approaches used in this thesis are listed below:

- Unexpected categories: gender is unknown; the length of the zip code is larger than 4, etc.
- Out of range: the values outside a permissible minimal or maximal range.
- Interquartile ranges: based on the assumption that deviation of statistical values should not be extreme. If we annotate Q_1 as 25% quartile, Q_3 as 75% quartile, and EVF as extreme value factor, outliers can be detected if $x > Q_3 + EVF \times (Q_3 - Q_1)$.

Format converting Some retrieved data elements cannot be recognized or handled directly, and a format converting step is needed. For example, date format data are converted to the month difference between the retrieved data and the current date; character strings are cataloged as nominal features. All this processing can be done by pieces of code in our system.

Missing values Incomplete data is an unavoidable problem in dealing with the real banking data. Data may be missed during many stages with different reasons: the customers do not provide full personal information intentionally, accidentally or by the concern for privacy; when entering the data into the IT system or manipulating data warehouse, mistakes might cause missing data as well; many customers do not own some products like saving account and credit card, the related information is apparently not applicable.

In this thesis, we use the most common imputation methods. For nominal features, the value of the feature that occurs most often within the same class is selected to be the value for all the unknown values of the feature. Sometimes we treat "unknown" itself as a new value for the features that contain missing values. For numeric features, substitute a feature's mean value computed from available samples belonging to the same class to fill in missing data values on the remaining cases.

It should be noted that the inapplicable items use zero as well as a real missing ("NA" or empty) label in some places in the dataset. For example, if customers do not own a saving account, the amount of monthly transaction of saving account is zero. In this case, we have to find the associated indicator flag and substitute the real missing item with the mean value for a numeric feature or "unknown" category for nominal features.

Normalization Normalization is important for neural networks and distance-based classifiers like kNN. When normalization is performed the value magnitudes are scaled to appreciably low values. In this thesis, we mainly use z-score normalization $\overline{x} = \frac{x-\mu}{\sigma}$.

Discretization Some classifiers like Naive Bayes require discrete values. Generally, discretization algorithms can be divided into unsupervised algorithms that discretize attributes without taking into account the class labels and supervised algorithms that discretize attributes by taking into account the class attribute. Liu et al. compared binning based approaches (unsupervised algorithm), entropy measure based approaches, dependency based approaches and accuracy based approaches (supervised algorithm) in [20], and the results were quite consistent and identified Ent-MDLP as the first choice to consider.

Following the suggestion, we use Ent-MDLP as discretization method in this thesis. The Ent-MDLP discretization process is illustrated in figure 2.6. First, the continuous values for a feature are sorted in either a descending or an ascending order; then, the cut-point candidates to split a range of continuous values are selected by entropy minimization; next, a minimum description length principle (MDLP) is used to determine if the candidate cut-point can be accepted: if the MDLP stop criterion as shown in formula 2.1 is satisfied, the discretization process stops, otherwise this procedure is continued. The formula of MDLP is

$$Gain(A, T; S) > \frac{log_2(N-1)}{N} + \frac{\Delta(A, T; S)}{N} \tag{2.1}$$

where $Gain(A,T;S)$ is information gain and $\Delta(A,T;S) = \log_2(3^k - 2) - [k\text{Ent}(S) - k_1\text{Ent}(S_1) - k_2\text{Ent}(S_2)]$, where Ent is entropy, k, k_1 and k_2 are the number of total class, left class and right class divided by the cut-point, respectively. The definition of entropy and information gain is show in formula 3.4 in section 3.5.

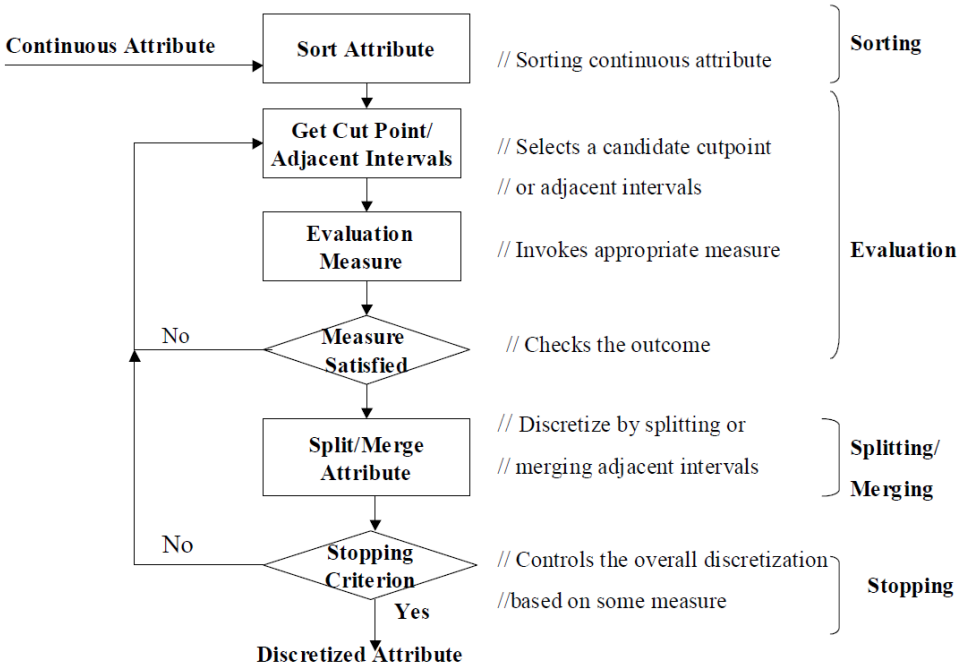


Figure 2.6: Discretization process [20].

Chapter 3

Classifiers in banking

3.1 A literature review of classifiers used in banking mortgage default field

A credit score is a numerical expression based on a statistical analysis of a person's credit files, to represent the creditworthiness of that person. The arrival of credit cards in the late 1960s made the banks and other credit card issuers realize the usefulness of credit scoring. Since the work of Altman in 1968 [21], who suggested using the so-called "Z score" to predict firms default risk, hundreds of research articles have studied this issue [22]. In the 1980s the success of credit scoring in credit cards meant that banks started using scoring for other products like mortgages [23]. At the beginning of using credit scoring, researchers focused on statistical or operational methods, including discriminant analysis, linear regression and linear programming. Gradually, more and more machine learning modelling approaches were imported into this field. Baesens et al. [22] reviewed some classification algorithms applied to eight real-life credit scoring data sets from major Benelux and UK financial institutions. Some well-known classification algorithms, e.g., k-nearest neighbour, neural networks, decision trees, support vector machines and least-squares support vector machines (LS-SVMs) were investigated. It was found that both the LS-SVM and neural network classifiers yield a very good performance, but also simple classifiers such as logistic regression and linear discriminant analysis perform very well for credit scoring. Feldman and Gross discussed the pros and cons of classification and regression trees (CART) in relation to traditional methods in [24]. They used CART to produce the first academic study of Israeli mortgages default data. Gan [25] investigated the risk management for residential mortgage in China and built an effective screening system to reduce the risk introduced by loan defaults. The paper reported an analytic study based on a real dataset of 641,988 observations provided by a Chinese commercial bank, and imported a profit matrix for the classification model to make the decision.

Behavioural score systems allow lenders to make better decisions in managing existing clients by forecasting their future performance. The decisions to be made include what credit limit to assign, whether to market new products to these particular clients, and if the account turns bad how to manage the recovery of the debt [23]. There are also plenty of research articles about behavioral score or behavioral assessment. Malik and Thomas [26] developed a Markov chain model based on behavioural scores to establish the credit risk of portfolios of consumer loans. The model is applied using data on a credit card portfolio from a major UK bank. Cumulative logistic regression was used to estimate the transition probabilities of the Markov chain. Hsieh used a self-organizing map neural network to identify groups of customers based on repayment behavior and recency, frequency, monetary behavioral scoring predictors in [27]. Case-based reasoning (CBR) is also a popular methodology for problem solving and decision-making in customer behavioral assessment [28, 29, 30]. Park proposed an analogical reasoning structure for feature weighting using a new framework called the analytic hierarchy process weighted k-NN algorithm in [31]. Krishnan [32] clustered the credit card debtors into homogeneous segments by using a self-organizing map, then developed credit prediction models to recognize the repayment patterns of each segment by using a Cox proportional hazard analysis. Ha [33] used a similar approach to estimate the expected time of credit recovery from delinquents.

The brief literature review above has shown several approaches, and naturally one question will pop up: "in these introduced or untouched approaches, which one should be used for our research problem?". It is not easy to tell which one is a winner. On the one hand, the research performed in this thesis is different from any work before and it should be the first application in short term recovery prediction of defaulted customers, therefore successful classification techniques in previous works cannot guarantee a global optimal solution to this problem. On the other hand, Capon [34] indicated that the use of credit scoring in the mortgage industry should be based on a pragmatic approach. The object is to predict who will recover accurately not to give explanation for why they recover or answer hypothesis on the relationship between default and other economic or social variables (at least the explanation is just icing on the cake, but not the fundamental requirement). This thesis will follow this pragmatic idea: **the most popular classification models will be tested and the best one will be deployed**.

3.2 Case based reasoning

Case based reasoning (CBR) solves new problems by using or adapting solutions that were used to solve old problems. A general CBR algorithm involves four steps [35]: (1) accepting a new problem representation, (2) retrieving relevant cases from a case base, (3) adapting retrieved cases to fit the problem at hand and generating the solution for it, and (4) store the problem in the case base and reuse it in future problem solving. This circle is illustrated in figure 3.1.

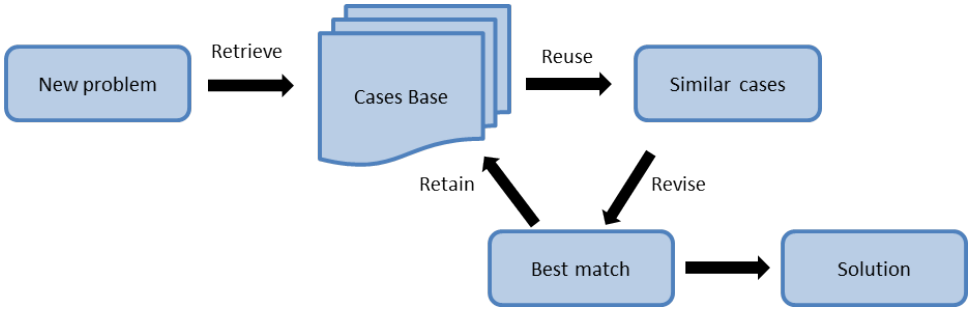


Figure 3.1: The CBR cycle.

The key issues in the CBR process are indexing and retrieving similar cases in the case base, measuring case similarity to match the best case, and adapting a similar solution to fit the new problem. Therefore, the measures of success of a CBR system depend on its ability to index cases and retrieve the most relevant ones in support of the solution to a new case. In this thesis, we use personal information to locate similar cases and the nearest neighbor of financial related data to match the best case as below:

- When retrieving candidates (similar cases), choose instances from the dataset with:
  - similar age (discretize age, and select the same category);
  - same gender;
  - same marriage status;
  - same geography location (The first two digits of the zip code) ;
  - in the new arrear customers who meet the four above requirements, select new arrear customers with the difference of mortgage monthly payment less than 0.1σ; if no customer's monthly payment falls in this area, enlarge the criteria to 0.2σ, 0.3σ, $\cdots$, until similar cases are found.
- To find the best match case:
  1. scale the financial related data by dividing them with monthly payment;
  2. normalize the financial related data;
  3. use wrapper feature selection method with kNN to select the top 10 related features;
  4. calculate the Euclidean distance[1], choose the best match with minimal distance from all candidates, and assign the label.

[1] A better way is using generalized matrix learning vector quantization to learn the relevant distance metric.

3.3 Logistic regression

Since it was first applied in customer credit behaviour prediction by Wiginton in 1980 [36], logistic regression has been widely used in the field of banking credit score. Logistic regression is an extension of linear regression. It has less restrictions on hypotheses about the data and can deal with qualitative indicators. The regression equation of LR is:

$$\ln(\frac{p_i}{1-p_i}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n, \tag{3.1}$$

where x_i is the i-th input variable, β_i are the regression coefficients, and the probability p_i obtained by Equation 3.1 is a bound of classification. The customer is considered a defaulter if it is larger than 0.5 or a delayer on the contrary. LR is proved as effective and accurate as LDA, but does not require input variables to follow a normal distribution [37].

A dummy variable takes the value 0 or 1 to indicate the absence or presence of some categorical effect. In a regression model, a dummy variable with a value of 0 will cause its coefficient to disappear from the equation. Conversely, the value of 1 causes the coefficient to function as a supplemental intercept, because of the identity property of multiplication by 1. So, encoding features as dummy variables allows easy interpretation, and it is a common method involved in studies of credit scoring.

For a nominal feature with C distinct categories, a set of C dummy variables can be generated.

$$x_1 = \begin{cases} 1 & \text{if the category is 1} \\ 0 & \text{otherwise} \end{cases}$$
$$x_2 = \begin{cases} 1 & \text{if the category is 2} \\ 0 & \text{otherwise} \end{cases}$$
$$\vdots$$
$$x_C = \begin{cases} 1 & \text{if the category is C} \\ 0 & \text{otherwise} \end{cases}$$

Since the C dummy variables are linearly dependent, any $C-1$ out of the C variables sufficiently identify a category. For numeric features, Ent-MDLP discretization is applied in order to convert numbers to categories; then dummy variables can be generated in the same way.

3.4 Naive Bayes

The Naive Bayes classifier is one of the oldest formal classification algorithms, and yet even in its simplest form it is often surprisingly effective. It is widely used in the statistical, data mining, machine learning, and pattern recognition communities [38].

Bayesian classifiers are statistical classifiers with a "white box" nature. Bayesian classification is based on Bayesian theory, which is described below. Consider a supervised learning problem in which we wish to approximate an unknown target function $f : X \to Y$, or equivalently $P(Y|X)$. After we apply Bayes rule, we see that $P(Y = y_i|x_1 \cdots x_n)$ can be represented as

$$P(Y = y_k|x_1 \cdots x_n) = \frac{P(X = x_1 \cdots x_n|Y = y_k)P(Y = y_k)}{\sum_j P(X = x_1 \cdots x_n|Y = y_j)P(Y = y_j)} \tag{3.2}$$

where y_k denotes the kth possible value for Y, x_i denotes the ith possible vector value for X, and where the summation in the denominator is over all legal values of the random variable Y [39].

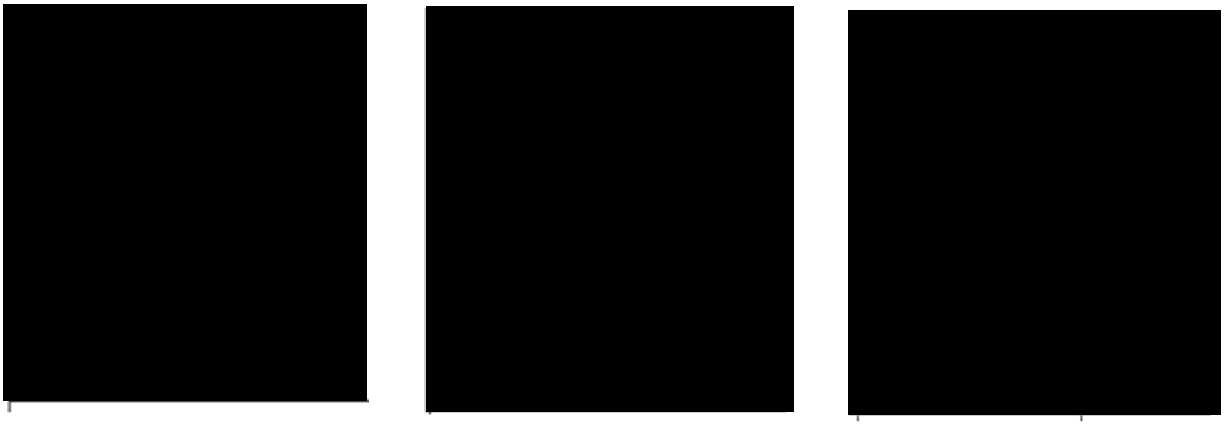
The Naive Bayes classifier assumes that the attributes of a sample are independent given the class. It reduces the number of parameters to be estimated dramatically when modelling $P(X = x_1 \cdots x_n|Y)$, i.e., we can have $P(X = x_1 \cdots x_n|Y) = \prod_{i=1}^{n} P(x_i|Y)$. Now use this to rewrite equation 3.2 as

$$P(Y = y_k|X = x_1 \cdots x_n) = \frac{P(Y = y_k)\prod_{i=1}^{n} P(x_i|Y = y_k)}{\sum_j P(Y = y_j)\prod_{i=1}^{n} P(x_i|Y = y_j))} \tag{3.3}$$

Because we are interested in the classification result, we assign the instance Y the label with the most probable value.

$$Y \leftarrow \arg\max_{y_k} P(Y = y_k) \prod_i P(x_i | Y = y_k)$$

For discrete features, we can simply substitute the nominal values into the Naive Bayes formula[2]. For continuous features, there are normally two approaches: estimating the probability density function or discretize the continuous variables. In our dataset, numeric features have different types of distributions as shown in figure 3.2: the amount of rest mortgages follows a normal distribution approximately, total number of successful cashless transactions over the past 12 months follows an edge peak right-skewed gamma distribution, and age has a bimodal distribution. So, we choose to employ the Ent-MDLP discretization method again to transform all features as nominal to make the system brief and clear.



(a) Histogram of total amount of rest mortgages

(b) Histogram of amount of cashless transactions

(c) Histogram of age

Figure 3.2: Histograms of some features in the dataset. Blue parts are histogram of defaulters and red parts are delayers.

Since the Naive Bayes classifier has a strong assumption about feature independence and there are indeed highly correlated features in the dataset (wealth class and incomes, the balance of bank account as 1st account holder and 2nd holder, etc), the correlated-based feature selection (CFS) method is applied before the Naive Bayes classifier. CFS evaluates subsets of features on the basis of the following hypothesis: "feature subsets contain features highly correlated with the classification, yet uncorrelated to each other" [40]. A greedy stepwise searching strategy is used with CFS to choose an independent subset.

3.5 Decision trees

The decision tree method is also known as recursive partitioning. It works as follows. First, according to a certain standard, the customer data are divided into limited subsets by using one (or more with linear split) attribute(s). Then the division process continues until the new subsets meet the requirements of the end node. The construction of a decision tree contains three elements: bifurcation rules, stopping rules and the rules deciding which class the end node belongs to. Bifurcation rules are used to divide new sub sets. Stopping rules determine whether the subset is an end node. In statistics and machine learning, there are several specific decision tree algorithms including ID3 (Iterative Dichotomiser 3), C4.5 (successor of ID3), CART (Classification And Regression Tree), CHAID (CHi-squared Automatic Interaction Detector), etc.

[2]Sometimes a smoothening step is needed for two reasons: to avoid zero-values; and to make the distribution smoother.

In this thesis C4.5 is employed. C4.5 uses information gain for its impurity function. Let S be a set, p be the fraction of defaulters, and q be the fraction of delayers. The definition of entropy is:

$$\text{Ent}(S) = -p\log_2(p) - q\log_2(q) \tag{3.4}$$

The entropy changes when we use a node in a decision tree to partition the training instances into smaller subsets. Information gain is a measure of this change in entropy. Suppose A is an attribute, S_v is the subset of S with $A = v$, Value(A) is the set of all possible values of A and $|S|$ is the size of set S, then

$$\text{Gain}(S, A) = \text{Ent}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \cdot \text{Ent}(S_v) \tag{3.5}$$

Besides information gain, differences between C4.5 and other variations of decision trees are listed as following [41]:

- Univariate splits: bifurcation rules only use one feature instead of a linear combination like $\alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 < c$.
- The number of branches could be larger than 2.
- Pruning: goes back through the tree once it has been created, and attempts to remove branches that do not help by replacing them with leaf nodes.

3.6 Summary

Until now, the framework and details of our classification system have been discussed. Equivalent to any other system: data gathering, feature selection, feature extraction and data preprocessing are first carried out, followed by various classifiers. We have not set up experiments in this chapter, because assessment metrics, one of the most critical steps especially in this imbalanced problem, have not been touched yet. In the next chapter, metrics, experiments and results of sampling techniques will be introduced.

Chapter 4

Sampling techniques in imbalanced learning

4.1 Nature of the problem

In this research problem, the two classes are imbalanced. Table 4.1 illustrates the number of defaulters and delayers from October 2012 to March 2013 and the ratio between defaulters and delayers is around 1 : 8. The reason of this imbalanced distribution is not complicated. According to the survey from the Arrears department, the vast majority of new arrear customers miss the first "incasso" just because their bank accounts are blocked temporarily, forget or feel lazy to pay the monthly debt. Most of them will repay before the end of the month.

Time	Oct 2012	Nov 2012	Dec 2012	Jan 2013	Feb 2013	Mar 2013
# New arrear customers	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
# Defaulters	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
# Delayers	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
percentage of defaulters	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

Table 4.1: Number of defaulters and delayers in 6 months.

This is a typical imbalanced learning problem. When classifying imbalanced data sets, most standard algorithms fail to properly represent the distributive characteristics of the data and result in unfavourable accuracies across the classes of the data [42]. Therefore, the imbalanced learning problem is warranting increasing exploration. Figure 4.1 illustrates the number of papers on imbalanced learning in data mining, pattern recognition and machine learning in the last decade in Google Scholar. As can be seen, the activity of publications in this field is growing fast.

Strictly speaking, any data set that exhibits an unequal distribution between its classes can be considered imbalanced. However, the common understanding in the community is that imbalanced data correspond to data sets exhibiting significant imbalances [42]. In some scenarios, one class severely over-represents another naturally, e.g., gene expressing data (100:1) [43] and shuttle system failure (1000:1) [44]. In some extreme cases the counter-examples are even absent. One-class classification, also known as unary classification, learns from a training set containing only the objects of that class. It is noted that imbalance also exists in multi-class classification problem [45], [46], [47]. In this thesis, we only focus on the binary class imbalanced learning problem. There are two reasons that imbalance falls into two cases: the data are naturally imbalanced, or it is too expensive to obtain data of the minority class. Obviously, customer behavior leads to our imbalanced research problem naturally.

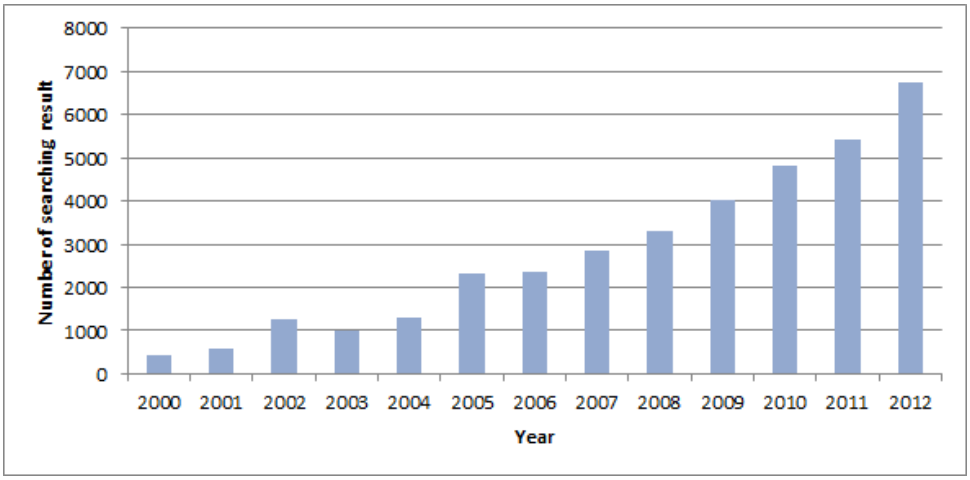


Figure 4.1: Number of results when searching publications with the keywords "imbalance learning" followed by "machine learning", "pattern recognition" or "data mining" in Google Scholar.

4.2 Assessment metrics

Assessment metrics should be discussed critically at the beginning of working on imbalanced learning, since inappropriate metrics would lead to wrong conclusions. In this section, we comparatively study the confusion matrix, singular metrics (*accuracy*, *error rate*, *precision*, *recall*, *F-measure*, *G-mean*) and curve metrics (*Receiver Operating Characteristic curve*) in imbalanced binary classification. In this thesis, **the ROC curve and the area under the curve (AUC) are the main metrics to compare different classifiers**.

4.2.1 Confusion matrix

Because defaulters are the principle component of risk cost which ING wants to cut down eagerly, we can rephrase our classification problem as "**can we detect defaulters from new arrear customers?**". Logically, we regard defaulters as the positive class and delayers as the negative class, then a representation of classification performance can be formulated by a confusion matrix, as illustrated in table 4.2.

		Predict class	
		Defaulter	Delayer
Actual class	Defaulter	true positive (TP)	false negative (FN)
	Delayer	false positive (FP)	true negative (TN)

Table 4.2: Confusion matrix for performance evaluation

4.2.2 Accuracy and error rate

Traditionally, the most frequent used metrics are *accuracy* and *error rate*.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \qquad ErrorRate = 1 - Accuracy$$

They provide a straightforward way of describing a classifier's performance on a given data set. However, for imbalanced data, especially strongly skewed data, using accuracy and error rate as measurement is not appropriate, because:

- One fact against the use of accuracy (or error rate) is that these metrics consider different classification errors to be equally important. However, highly imbalanced problems generally have highly

non-uniform error costs that favor the minority class, which is often the class of primary interest [48]. In our case, a defaulter misclassified as delayer is less acceptable than a delayer labelled as defaulter, since the misclassified defaulters will lose the opportunity to be contacted by case managers.

- Accuracy (or error rate) will lead to unexpected conclusions. For example, table 4.3a shows one possible output by decision trees. The confusion matrix is biased to delayers severely and gets an accuracy value of 90%. Another even more extreme example in table 4.3b is a naive classifier which regards every new arrear customer as delayer. The accuracy is also 90%, which sounds as excellent as the classifier in 4.3a!

		Predict class	
		Defaulter	Delayer
Actual class	Defaulter	100	900
	Delayer	100	8900

(a) One possible output of a decision tree.

		Predict class	
		Defaulter	Delayer
Actual class	Defaulter	0	1000
	Delayer	0	9000

(b) Naive classifier: assigns all customers as delayers.

Table 4.3: Confusion matrix of two classifiers with accuracies of 90%.

4.2.3 Singular assessment metrics

From the confusion matrix, we can easily extend the definition of accuracy and error rate to two metrics which measure the classification performance on the positive and negative class independently:

True positive rate $TP_{rate} = \frac{TP}{TP+FN}$ is the percentage of defaulters correctly classified as defaulters;

True negative rate $TN_{rate} = \frac{TN}{FP+TN}$ is the percentage of delayers correctly classified as delayers.

TP_{rate} and TN_{rate} just measure completeness (i.e., how many examples of the positive class were labelled correctly). Exactness (i.e., of the examples labelled as positive, how many are actually labeled correctly) should be also paid close attention. So, positive predictive value (ppv) and negative predictive value (npv) are imported as well.

Positive predictive value $ppv = \frac{TP}{TP+FP}$ is the proportion of the true defaulters against all the predicted defaulters;

Negative predictive value $npv = \frac{TN}{TN+FN}$ is the proportion of the true delayers against all the predicted delayers.

There are different names describing these metrics in different fields. For example, in information retrieval, TP_{rate} is called *recall* and positive predictive value is *precision*. In statistic test theory, *hit rate* and $false\ alarm\ rate$ are true positive rate and false positive rate, respectively. *Sensitivity* and $specificity$ are also common terminology in statistical binary tests, sensitivity is true positive rate and specificity is true negative rate. In this thesis, precision and recall are used.

Intuitively, the main goal for learning from imbalanced new arrear customers in arrears is to improve the recall without hurting the precision. However, recall and precision goals can be often conflicting, since when increasing the true positive rate for the defaulters, the number of false positives (i.e., delayers misclassified as defaulter) can also be increased; this will reduce the precision. So, here comes a question: suppose two confusion matrices are given like in table 4.4, how do we decide which confusion matrix is better than the other when one has higher recall while another has higher precision?

		Predict class	
		Defaulter	Delayer
Actual class	Defaulter	600	400
	Delayer	1400	7600

(a) Recall is 60% and precision is 30%

		Predict class	
		Defaulter	Delayer
Actual class	Defaulter	500	500
	Delayer	500	8500

(b) Recall is 50% and precision is 50%

Table 4.4: How to judge which confusion matrix is better?

Two measures are frequently adopted in the research community to provide comprehensive assessments of imbalanced learning problems:

F-Measure $= \frac{(1+\beta^2) \cdot Recall \cdot Precision}{\beta^2 \cdot Recall + Precision}$, where β is a coefficient to adjust the relative importance of precision versus recall (usually, $\beta = 1$). The F-measure score can be interpreted as a weighted average of the precision and recall, where an F-measure reaches its best value at 1 and worst score at 0. It is easy to see that the F-measure of table 4.4a is $(2 \cdot 0.6 \cdot 0.3)/(0.6 + 0.3) = 0.4$, while it is $(2 \cdot 0.5 \cdot 0.5)/(0.5 + 0.5) = 0.5$ in table 4.4b.

G-mean $= \sqrt{\frac{TP}{TP+FN} \times \frac{TN}{TN+FP}}$. G-mean indicates the balance between classification performances on the majority and minority class. This metric takes into account both true positive rate and true negative rate. Again, given table 4.4, we get G-mean of (a) is $\sqrt{0.6 \cdot 0.8444} = 0.711$ and G-mean of (b) is $\sqrt{0.5 \cdot 0.9444} = 0.687$.

Though F-Measure and G-Mean are great improvements over accuracy, they are still ineffective in answering more generic questions about classification evaluations. In the next section, curve metrics will be introduced, which can give answers of assessing the holistic classification performance.

4.2.4 Curve metrics

Some singular metrics were introduced in the previous section. If we plot one singular metric against another in a two-dimensional graph, more meaning can be represented. One most commonly used graph is the ROC graph, which plots TP rate on the Y axis and FP rate on the X axis. Before discussing the ROC graph, we first compare the hard-type classifiers and soft-type classifiers.

Hard-type classifiers and soft-type classifiers Many classifiers, such as nearest neighbor or decision trees, are designed to produce only a class decision, i.e., a label of defaulter or delayer on each new arrear customer. When such classifier is applied to a test set, it yields a single confusion matrix, which in turn corresponds to one (FP rate,TP rate) pair. They are called hard-type classifiers or discrete classifiers.

Some classifiers, such as Naive Bayes or neural networks, naturally yield an instance a numeric value that represents the degree to which an instance is a member of a class. Normally, the numeric outputs could possess one of three kinds of meaning below:

- Strict probabilities: the outputs adhere to standard theorems of probability, like Naive Bayes or Multi-variance discriminant analysis.
- Relative probabilities: some bagging based classifiers like random forests use voting to yield the final output. The outputs are kinds of pseudo probabilities.
- General and uncalibrated scores: the only property that holds is that a higher score indicates a higher tendency to predict the positive class, like logistic regression.

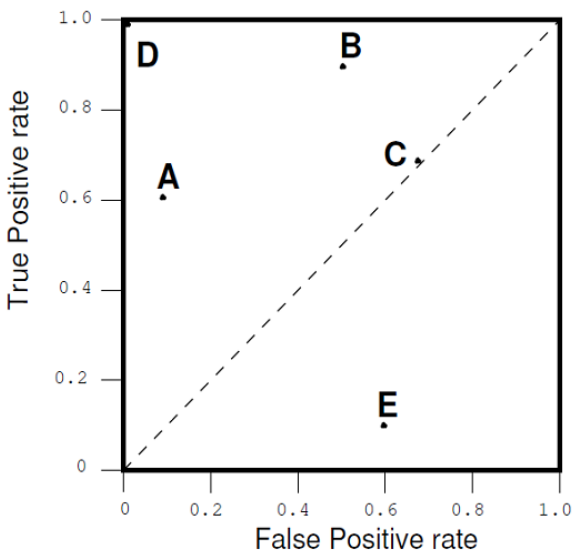
We call these three types of classifiers soft-type classifiers or probabilistic classifiers. Soft-type classifiers can be used with a threshold to produce a discrete classifier: if the classifier output is above the threshold, the classifier produces a defaulter, else a delayer.

Fawcett listed several methods which make discrete classifiers generate scores rather than just a class label in [49]. For example, a decision tree determines a class label of a leaf node from the proportion of instances at the node, and these class proportions may serve as a score [50]; MetaCost employs bagging to generate an ensemble of discrete classifiers, each of which produces a vote, and the set of votes could be used to generate a score [51].

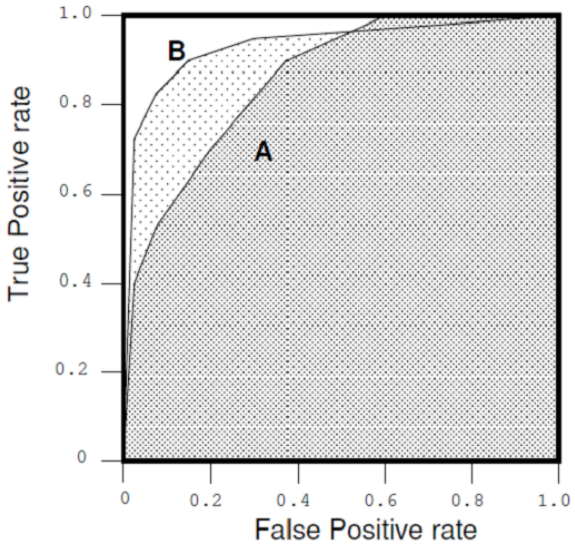
ROC graph As aforementioned, a discrete classifier only generates one (FP rate, TP rate) pair. In other words, one discrete classifier corresponds to one single point in ROC space. Figure 4.2a shows an ROC graph with five typical classifiers labeled A through E.

Before discussing points A to E, let us look at several special points in the ROC graph. Both points $(0, 0)$ and $(1, 1)$ are two naive approaches: $(0, 0)$ represents an all-negative classifier, which assigns all new arrear customers as delayer; $(1, 1)$ represents an all-positive classifier, which labels new arrear customers as defaulter unconditionally. Any point on the diagonal line represents the strategy of randomly guessing a class. For example, if a classifier assigns the label by tossing a coin, it can be sure that it gets half defaulters and half delayers correct; this yields the point $(0.5, 0.5)$ in ROC space. If it guesses the positive class 70% of the time, it can be expected to get 70% of the positives correct but its false positive rate will increase to 70% as well, yielding point C in figure 4.2a.

From the ROC graph it is easily concluded that one point is better than another if it is to the northwest (TP rate is higher, FP rate is lower, or both) of the other. So, points A and B are better than C, while point D $(0, 1)$ represents the perfect classification. Note that a point which is southeast to the diagonal line does not mean that the classifier cannot provide useful information. On the contrary, the classifier is informative but just misused. For instance, if we reverse the classification result of point E, then it will just produce point A in figure 4.2a.



(a) A basic ROC graph showing five discrete classifiers [49]: C is a random guessing classifier; A and B outperform C; E is symmetric with A; D is the perfect classifier.



(b) Classifier B is generally better than A except at $FPrate > 0.6$ where A has a slight advantage. But in practice the AUC performs very well and is often used when a general measure of predictiveness is desired [49].

Figure 4.2

ROC Curve and Area under the curve (AUC) In the case of soft-type classifiers, a threshold can divide the output into two parts: above threshold are assigned positive labels, else negative labels. In other words, a threshold produces a discrete classifier and one point on the ROC graph. If varying the threshold between the minimal probability (score) and maximal probability (score), a series of points are generated on the ROC graph. Changing the threshold value corresponds to moving from one point to

another point, and by traversing all thresholds, an ROC curve is generated. Even for discrete classifiers, it is straightforward to have them produce soft-type outputs as aforementioned, and generate ROC curves in the same way.

Similar with the evaluation criterion comparing two points in the ROC graph, the ROC curve which protrudes to the northwest corner outperforms ROC curves under it. The area under the curve, abbreviated AUC, is a common method to convert the ROC curve to a single scalar value representing performance. The AUC value is always between 0 and 1. As the analysis of random guessing in the previous paragraph, any ROC curve having AUC lower than 0.5 is misused: just simply getting the complement value of the output will give an AUC above 0.5 again.

Although it is possible for a high-AUC classifier to perform worse in a specific region of ROC space than a low-AUC classifier as illustrated in figure 4.2b, a classifier with higher AUC normally has better average performance.

4.3 The state-of-the-art for imbalanced learning

Nowadays, there is a common understanding in the community that most traditional machine learning methods are affected by imbalanced distributed data [52, 53, 54, 55]. One example was given by Lemnaru and Potolea in [55]: the experimental result on 32 data sets from the UCI machine learning data repository showed that decision trees and SVMs perform strongly worse when the data is imbalanced. That is mainly because they are not built with the purpose of this domain: first, they are designed to maximize accuracy, which has already been proved to be an improper metric in the last section; secondly, the generated model, pattern or rules that describe the minority concepts are often rarer and weaker than those of majority concepts, since the minority class is often both outnumbered and underrepresented.

The remedies to deal with the problem of class imbalance are of three different levels according to the phases in learning, i.e., data level methods for handling imbalance, which contain changing class distributions mainly by re-sampling techniques and feature selection in the feature level, classifiers level by manipulating classifiers internally and ensemble learning level [56]. Against imbalanced data distributions, changing class distributions is the most natural solution. Sampling methods seem to be the dominate type of approach in the machine learning community as the way these methods work is straightforward [57]. The following paragraphs will mainly focus on the sampling methods, such as random oversampling and undersampling, informed undersampling, synthetic sampling with data generation.

Random sampling The basic sampling methods include random undersampling and random oversampling. Like the literal meaning of "random sampling", undersampling eliminates majority-class examples randomly while oversampling duplicates minority-class examples randomly. Both of these sampling techniques decrease the overall level of class imbalance, thereby making the rare class less rare.

These random sampling methods have several drawbacks. Intuitively, undersampling discards potentially useful majority-class examples and thus can degrade classifier performance. With regards to oversampling, because it introduces additional training cases, the training time increases when building a classifier. Worse yet, because oversampling makes exact copies of examples, it may lead to overfitting: although the training accuracy is high, the classification performance on the unseen testing data is generally far worse [58]. Some studies have shown simple oversampling to be ineffective at improving recognition of the minority class and why undersampling may be a better choice [59].

Informed undersampling Since random undersampling would miss potentially useful information, Zhang and Mani proposed four undersampling methods combined with k-nearest neighbor in [60], which only keep useful majority-class instances. The basic idea behind them is that the majority-class instances which are surrounded by minority-class instances are more likely to locate around a decision boundary, which is slightly similar with the theory of Support Vector Machines. The four proposed methods are called NearMiss-1, NearMiss-2, NearMiss-3, and the "most distant" method. The NearMiss-1 method selects those majority examples whose average distance to the three closest minority class examples is the smallest to keep, while the NearMiss-2 method selects the majority class examples whose average distance to the three farthest minority class examples is the smallest. NearMiss-3 selects a given number of the closest majority examples for each minority example to guarantee that every minority example is

surrounded by some majority examples. Finally, the "most distant" method selects the majority class examples whose average distance to the three closest minority class examples is the largest.

There are some other informed undersampling methods such as EasyEnsemble and BalanceCascade algorithms [61], which will be discussed in chapter 4.

Synthetic sampling with data generation Synthetic sampling with data generation techniques have also attracted much attention. The synthetic minority oversampling technique (SMOTE) algorithm is the most enlightened approach, which oversamples by introducing new, non-replicated minority class examples [62]. Its main idea is that minority class examples are generated by interpolating examples from the line segments that join the k minority-class nearest neighbors. For every minority instance, its k nearest neighbors of the same class are calculated based on euclidean distance, then some examples are randomly selected from them according to the oversampling rate. After that, randomly select one of the k nearest neighbor, then new synthetic examples are generated along the line between the minority example and its selected nearest neighbors. Thus, the overfitting problem is avoided and causes the decision boundaries for the minority class to spread further into the majority class space. Figure 4.3 shows an example of the SMOTE procedure.

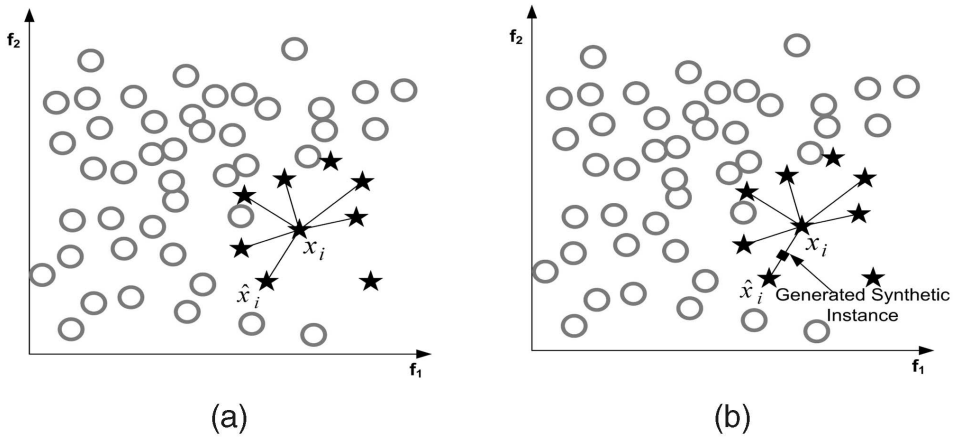


Figure 4.3: (a) Example of the K-nearest neighbors for the x_i example under consideration ($k = 6$). (b) Data creation based on euclidian distance. [42]

Some approaches were proposed to improve the SMOTE algorithm, such as Borderline-SMOTE [63], SMOTE with ENN [48], SMOTE with Tomek links [48], SMOTE-RSB [64] and SMOTEBoost [65]. Here follows a short introduction of Borderline-SMOTE. Suppose among k nearest neighbors of minority class instance X_i, m neighbors belong to the minority class and $k - m$ ones belong to the majority class. X_i can be regarded as "*SAFE*", "*DANGEROUS*" or "*NOISY*" according to the distribution of m and $k - m$ as below:

- "*SAFE*": $m > k - m$
- "*DANGEROUS*": $0 < m <= k - m$
- "*NOISY*": $m = 0$

Since the examples in "*DANGEROUS*" represent the borderline minority class examples (the examples that are most likely to be misclassified), the "*DANGEROUS*" set is input to the SMOTE algorithm. Figure 4.4 illustrates an example of the Borderline-SMOTE procedure. We see that Borderline-SMOTE only generates synthetic instances for those minority examples closest to the border.

4.4 Experimental study and discussion

Until now, we introduced the dataset, the classification pipeline and the classifiers in chapter 2 and 3, then explained the imbalanced distribution of our dataset, the evaluation metrics and the approaches of imbalanced learning in the earlier sections in this chapter. In this section, we carry out the empirical

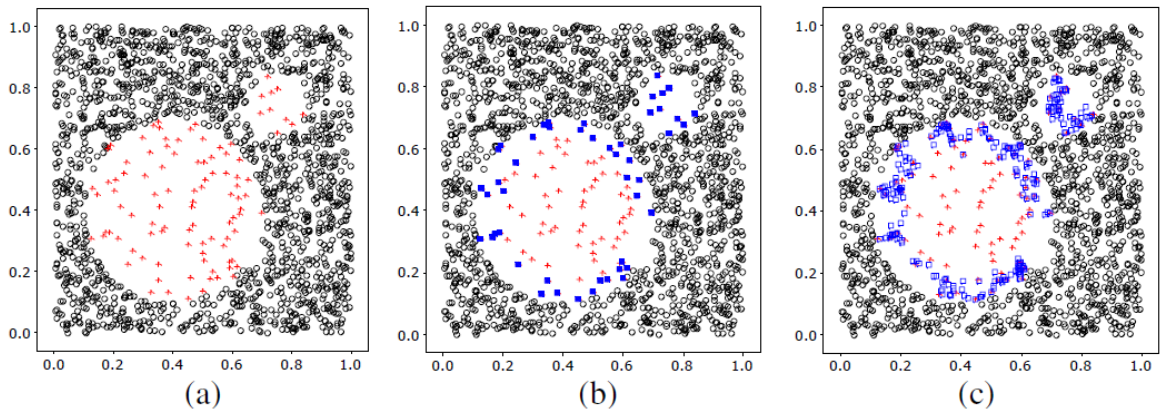


Figure 4.4: (a) The original distribution of the Circle data set. (b) The borderline minority examples (solid squares). (c) The borderline synthetic minority examples (hollow squares). [63]

comparison of the algorithms that we have reviewed in section 3, i.e., case based reasoning, Naive Bayes, decision trees (C4.5) and logistic regression. Then, three types of sampling methods, random undersampling, random oversampling and SMOTE are employed here to comparatively study the impact of sampling techniques on the classification performances. True positive rate, precision of defaulters, True negative rate, F-score, AUC and ROC curve are employed as evaluation metrics. The normal k-fold cross validation would bring the situation that the distribution of defaulters and delayers in each fold are slightly different. In order to reduce the deviation of the result, **stratified k-fold cross validation** is adopted to ensure the numbers of instances in both majority and minority class are strictly equal in each fold. Following the common usage, we use 10 folds in our experiments. As mentioned in chapter 2, the total dataset has around [redacted] new arrear customers, and such a large number of instances will lead to remarkable computation time, especially with 10-fold cross validation. A randomly sampled subset ([redacted]) will be used in the phase of comparing classifiers for the purpose of reducing experiment time. When the best classifier is spotted, the whole dataset will be employed again to build the final model.

Our aim is to answer several questions in the scenario of this arrears customers imbalanced classification problem:

- What is the impact of the imbalanced data distribution?
- How do sampling techniques improve the performance?
- Find the best solution to classify defaulters and delayers so far

Table 4.6 lists all performance indicators with average value and standard deviation of the result of 10-fold cross validation. In order to explain the results clearly, we extract parts of the data from table 4.6 to form some new table and figures. **CBR**, **NB**, **DT** and **LR** in the following tables and figures in this and the rest of the chapters are abbreviations for Case-based Reasoning, Naive Bayes, Decision tree, logistic regression.

Basic classifiers We first consider the results of singular assessment metrics without sampling techniques as shown in table 4.5. The first four columns list the four numbers of the confusion matrices. In order to observe easily, the numbers are rounded as the unit of tens or hundreds. The next four columns in the table show true positive rate (the percentage of defaulters correctly classified as defaulters), positive predictive value (the proportion of the true defaulters against all the predicted defaulters), true negative rate (the percentage of delayers correctly classified as delayers) and negative predictive value (the proportion of the true delayers against all the predicted delayers), respectively.

It is clear that none of the four classifiers can obtain good classification performance for both defaulters and delayers simultaneously. Case-based reasoning, decision trees and logistic regression can distinguish most delayers (TN rates are aound 98%) but the ability to detect defaulters is very poor (TP rates

are less than 12%). That is because delayers dominate the dataset and all the instances are equally weighted: Case based reasoning has a higher probability to find delayers than nearest neighbors; the splitting rule of a Decision tree can almost ignore defaulters since information gain can get benefit from a good splitting of delayers; the aim of logistic regression is to optimize the cost function, i.e, the sum of square errors of all instances, and biasing to the majority class can obtain smaller cost.

If we just employed the singular metrics in table 4.5, comparing the performances of the four classifiers is impossible because we do not know how to trade-off the confusion matrix.

Methods	TP	FN	FP	TN	TP rate	ppv	TN rate	npv
CBR					0.0576	0.3884	0.9962	0.9089
NB					0.9112	0.1064	0.1971	0.9542
DT					0.1141	0.4181	0.9830	0.9134
LR					0.0509	0.4584	0.9935	0.9087

Table 4.5: Using singular metrics to assess basic classifiers. TP, FN, FP, TN, TP rate, ppv, TN rate and npv are abbreviations for true positive, false negative, false positive, true negative, true positive rate, positive predictive value, true negative rate, negative predictive value. All these results are achieved when the cut-off threshold is **0.5**, which is the default threshold of these classification methods.

Sampling techniques Next, we evaluate the impact of sampling techniques by singular metrics. By observing the TP rate and TN rate with and without a sampling technique, we can see clearly that sampling generates more balanced results, that makes the value of one metric higher and another value lower.

The F-measure was introduced in subsection 4.2.3. It is a weighted average of the true positive rate and precision, where an F-measure reaches its best value at 1 and its worst score at 0. Figure 4.5 plots the F-measure of four classifiers with four sampling techniques. For each group of bars, it is clear that the performance of the original classifier (the leftmost white bar) is significantly lower than the performance of classifiers with sampling techniques (other three bars). Put another way, the detection of defaulters becomes more accurate and more effective.

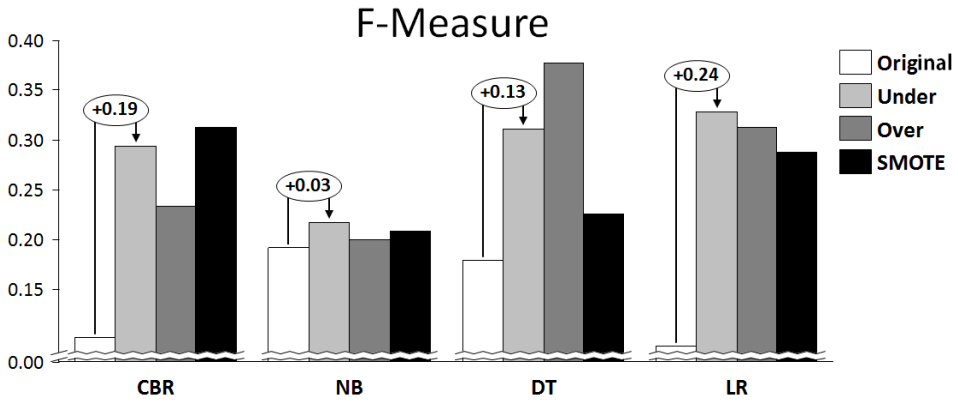


Figure 4.5: Bar charts of F-measure. Original, Under, Over and SMOTE in the legend stand for basic classifier, randomly undersampling, randomly oversampling and SMOTE, respectively.

Comparison by using ROC curve and AUC As discussed in section 4.2.4, curve metrics assess the performance of classifiers holistically. Figure 4.6 compares AUC values of four classifiers with four kinds of sampling techniques. All four groups of bars indicate that undersampling (the second left bar) outperforms the original classifiers (the left most bar) significantly. If comparing with random oversampling and SMOTE, undersampling performs also better or equally (the AUC of logistic regression with random oversampling is close to undersampling).

Another interesting result is that SMOTE does not lift the performance remarkably. The testing result of SMOTE on Naive Bayes and logistic regression even decreases slightly. We can get explanation from the study of SMOTE for high-dimensional class-imbalanced data by Blagus and Lusa in [66]. On high dimensional and imbalanced data like our case, they set up a group of experiments and conclude that SMOTE has hardly any effect on most classifiers trained on high-dimensional data. KNN-based feature selection before SMOTE can eliminate this problem. In our experiment, however, we do not use KNN-based feature selection[1], hence the SMOTE oversampling does not work very well.

Figure 4.7 plots ROC curves of four classifiers with undersampling. The ROC curve of logistic regression (black curve) dominates the other three curves in the whole range of the graph. A student t-test also shows AUC of logistic regression exceeds the other three significantly. The p-value of the t-test between logistic regression with undersampling and Decision tree with undersampling is **0.006455**.

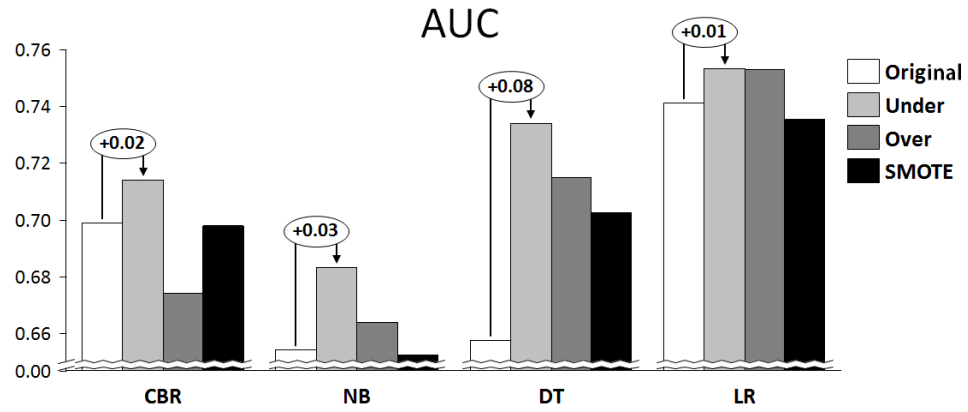


Figure 4.6: Bar charts of AUC. Original, Under, Over and SMOTE in the legend stand for basic classifier, randomly undersampling, randomly oversampling and SMOTE, respectively.

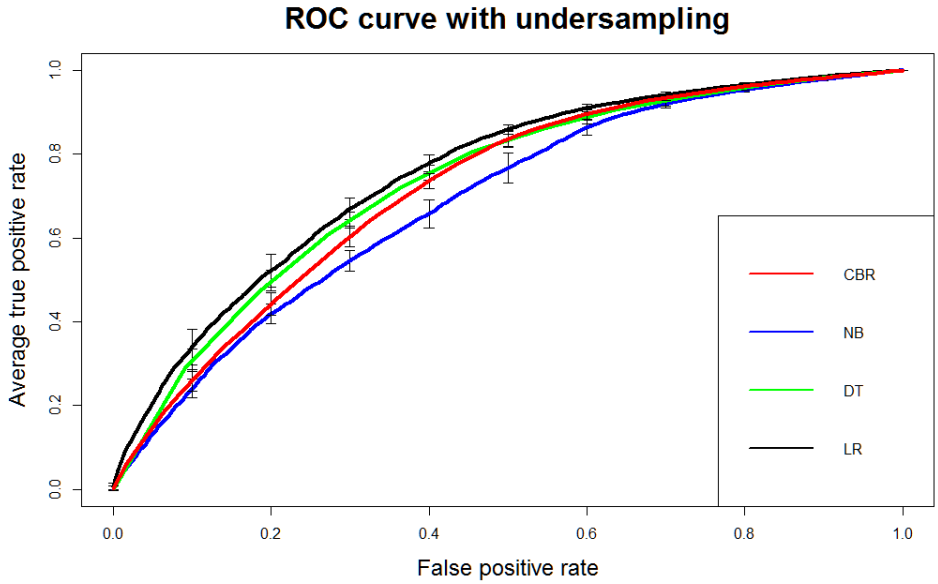


Figure 4.7: ROC curves of case-based reasoning, Naive Bayes, Decision tree and logistic regression with undersampling. The bars attached on the curves are standard deviation bars.

[1]To be precise, we use the Naive-Bayes-based wrapper method to select the features when we test the Naive Bayes classifier, and logistic-regression based wrapper method for the testing of the logistic regression classifier.

Three naive approaches As reference, three naive approaches are also tested here as shown in table 4.6. The performance of all three naive classifiers is very poor, but it is noted that if we use accuracy as assessment metric, "All delayers" can even get 90.5% accuracy and beats all other four classifiers! That also confirms that accuracy is not a good metric for imbalanced data.

4.5 Summary

In this chapter, we first described the situation of imbalanced distributed data, then assessment metrics and some sampling techniques were introduced. At last, 10-fold cross validation was applied with four classifiers and sampling techniques. The empirical study on the testing results can answer the three questions in the previous section:

What is the impact of the imbalanced data distribution: imbalanced data cause the basic classifiers to bias to the majority class seriously.

How do sampling techniques improve the performance: sampling makes performance more balanced and increases AUC. Random undersampling works better than the other three techniques.

Find the best solution to classify defaulters and delayers: logistic regression with undersampling is the best classifier so far. It gives an AUC of 0.7531 and outperforms other tested classifiers significantly. According to the literature review, we have already realized that logistic regression is one of the most popular approaches in the field of credit score prediction. Our testing result confirms the success of logistic regression. Bolton indicated in [67] that logistic regression is the most favored method in practice of credit score prediction due to (almost) no assumptions imposed on variables, with the exception of missing values and multicollinearity among variables.

In the next chapter, ensemble methods will be used to learn models and we will show that they achieve better performance in classifying defaulters and delayers.

Methods	Recall	Precision	TN rate	F-measure	G-means	ROC Area
All defaulters	1	0.095	0	0.1735	0	NA
All delayers	0	0	1	0	0	NA
Random (coin toss)	0.5	0.095	0.5	0.1596	0.5	NA
CBR	0.0576 ± 0.110	0.3884 ± 0.066	0.9962 ± 0.012	0.1005 ± 0.031	0.2377 ± 0.051	0.6989 ± 0.018
CBR + undersampling (1:1)	0.5959 ± 0.018	0.1942 ± 0.120	0.7418 ± 0.056	0.2928 ± 0.068	0.3397 ± 0.011	0.7140 ± 0.050
CBR + oversampling (1:1)	0.6832 ± 0.028	0.1403 ± 0.007	0.5609 ± 0.009	0.2327 ± 0.009	0.3095 ± 0.011	0.6742 ± 0.015
CBR + SMOTE (1:1)	0.3272 ± 0.025	0.3016 ± 0.018	0.9203 ± 0.008	0.3128 ± 0.011	0.3136 ± 0.011	0.6977 ± 0.010
NB	0.2092 ± 0.029	0.3964 ± 0.002	0.9671 ± 0.038	0.3966 ± 0.004	0.4498 ± 0.030	0.6540 ± 0.012
NB + undersampling (1:1)	0.3330 ± 0.106	0.3543 ± 0.017	0.9362 ± 0.163	0.3544 ± 0.021	0.5586 ± 0.077	0.6830 ± 0.023
NB + oversampling (1:1)	0.8462 ± 0.047	0.1132 ± 0.009	0.3003 ± 0.069	0.1993± 0.013	0.3089 ± 0.009	0.6638 ± 0.009
NB + SMOTE (1:1)	0.6911 ± 0.049	0.1229 ± 0.011	0.4788 ± 0.062	0.2081 ± 0.014	0.2907 ± 0.009	0.6521 ± 0.019
DT	0.1141 ± 0.017	0.4181 ± 0.062	0.9830 ± 0.004	0.1785 ± 0.024	0.3340 ± 0.025	0.6574 ± 0.018
DT + under-sampling (1:1)	0.6704 ± 0.027	0.2026 ± 0.012	0.7222 ± 0.022	0.3110 ± 0.015	0.6956 ± 0.014	0.7339 ± 0.008
DT + oversampling (1:1)	0.5787 ± 0.016	0.2805 ± 0.012	0.8442 ± 0.005	0.3777 ± 0.013	0.4028 ± 0.012	0.7147 ± 0.009
DT + SMOTE (1:1)	0.2008 ± 0.047	0.2636 ± 0.027	0.9414 ± 0.011	0.2258 ± 0.035	0.2290 ± 0.034	0.7023 ± 0.049
LR	0.0509 ± 0.007	0.4584 ± 0.076	0.9935 ± 0.002	0.0915 ± 0.012	0.2245 ± 0.015	0.7412 ± 0.017
LR + under-sampling (1:1)	0.6069 ± 0.032	0.2242 ± 0.014	0.7793 ± 0.010	0.3273 ± 0.019	0.6876 ± 0.020	0.7531 ± 0.017
LR + oversampling (1:1)	0.6480 ± 0.034	0.2069 ± 0.017	0.7381 ± 0.029	0.3130 ± 0.018	0.3656 ± 0.014	0.7529 ± 0.013
LR + SMOTE (1:1)	0.3260 ± 0.115	0.2825 ± 0.053	0.8935 ± 0.110	0.2873 ± 0.026	0.2947 ± 0.017	0.7354 ± 0.029

Table 4.6: Testing results. CBR, NB, DT and LR are abbreviations for case-based reasoning, Naive Bayes, decision trees, logistic regression. The recall, precision, TN rate, F-measure and G-means are achieved when the cut-off threshold is **0.5**, which is the default threshold of these classification methods.

Chapter 5

Ensemble methods on imbalanced data

5.1 Ensemble learning

The idea of ensemble learning is to build a prediction model by combining the strengths of a collection of simpler base models. The earliest study of ensemble learning is from Dasarathy and Sheela [68] in 1979, which discussed partitioning the feature space using two or more classifiers. Since the huge success of bagging [69], Random forests [70], random subspace methods [71], boosting [72], AdaBoost [73] and Gradient boosting [74], ensemble learning is one of the hottest research fields in the machine learning community. The following subsections will introduce these ensemble methods successively.

5.1.1 Bagging

The Bagging algorithm (**B**ootstrap **agg**regat**ing**), proposed by Breiman in [69], is one of the earliest ensemble learning algorithms. Following the abbreviation, Bagging is based on the concepts of bootstrapping and aggregating, and incorporates the benefits of both approaches.

Given a training set S of size N, a Bootstrap sample B is obtained by uniformly subsampling N' instances from S with replacement. By uniformly sampling with replacement, some training examples may be repeated in a bootstrap sample, while some instances may not appear. The probability which an instance has in the bootstrap sample is $1-(1-1/N)^{N'}$. If $N'=N$ and for large N, the probability becomes $\lim_{N\to\infty} 1-(1-1/N)^N = 1-1/e \approx 0.632$. In other words, one bootstrap sample is expected to have 63.2% of the unique examples of S.

M bootstrap samples B_1, B_2, $\cdots$, B_M are generated. Each sample B_i is used to train a different base learner C_i of the same type. Then, aggregating actually means combining classifiers by majority voting. The final classifier C^* is built from C_1, C_2, $\cdots$, C_M whose output is the class predicted most often by its base classifiers. Algorithm 1 shows the implementation of bagging.

Algorithm 1 Bagging

Input: training set S, Inducer I, integer M (number of bootstrap samples)

for $i = 1$ to M **do**
  $S' =$ bootstrap sample from S (i.i.d. sample with replacement)
  $C_i = I(S')$
end for
$C^*(x) = \arg\max_{y\in Y} \sum_{i:C_i(x)=y} 1$ (the most often predicted label y)

Output: classifier C^*

Simple as it is, a combined classifier gives better results than individual classifiers, because this strategy can reduce variance when combined with bootstrap sampling. The detailed explanation can be found in the original paper [69]. It is noteworthy that relatively unstable base classifiers (e.g. Decision trees and neural networks) should be used so that sufficiently different decision boundaries can be obtained for small perturbations in different bootstrap samples. Stable algorithms (e.g. k-nearest neighbors and Linear Discriminant Analysis) cannot get significant benefit or even may slightly degrade the performance, because stable learning methods have low variance to begin with, and bagging may not help much.

5.1.2 Random forests

Like the literal meaning, a random forest [70] is a "forest", which aggregates unpruned classification or regression trees. The trees are induced from bootstrap samples of the training data by using random feature selection in the tree induction process. The final prediction is made by aggregating (majority vote for classification or averaging for regression) the predictions of the ensemble. Algorithm 2 shows the procedure.

Algorithm 2 Random forests for Regression or Classification

Input: training set S, integer M (number of bootstrap samples)

1. For $b = 1$ to M:
   (a) Draw a bootstrap sample $\boldsymbol{B}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to fit the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^M$.

To make a prediction at a new point x:

Regression: $\widetilde{f}_{\mathrm{rf}}^M(x) = \frac{1}{M}\sum_{b=1}^M T_b(x)$

Classification: Let $\widetilde{C_b}(x)$ be the class prediction of the bth random-forest tree. Then $\widetilde{C}_{\mathrm{rf}}^M(x) = majority\ vote\{\widetilde{C}_b(x)\}_1^M$

As we discussed in the previous subsection, the idea in bagging is to aggregate many noisy but approximately unbiased models, and hence reduce the variance [70]. Trees are ideal candidates for bagging. The idea in random forests is to improve the variance reduction of bagging by reducing the correlation between the trees, without increasing the bias too much. This is achieved in the tree-growing process through random selection of the input variables.

During the phase of training, an estimation of the error rate can be obtained: at each bootstrap iteration, some data not in the bootstrap sample are called "out-of-bag", or OOB. Each data instance would be out-of-bag around 36% of the time. First, predict each OOB by the corresponding tree. Then aggregate the OOB predictions and calculate the error rate. Liaw and Wiener [75] indicated that the OOB estimate of the error rate is quite accurate, so it can be usually used to tune parameters of random forests.

The number of trees N and the number of randomly selected features m_{try} are the two most important factors to tune. The number of trees necessary for good performance normally grows with the number of available features. Experiments can decide how many trees are necessary. We can stop building trees, when the AUC value does not increase anymore, or in another way, the subsets of the forest works as well as the full forest. For selecting the number of features m_{try}, the default value is $p/3$ for regression problems or $\sqrt{p}$ for classification problems, where p is the number of features. Breiman recommends trying the default value, half of the default, and twice the default, and pick the best. However, Liaw

and Wiener pointed out in [75] that the way of searching the best m_{try} recommended by Breiman is not always the best. Even $m_{try} = 1$ can give very good performance for some data, and if one has a very large number of variables but expects only very few to be "important", using larger m_{try} may give better performance. So, in our case we will test and select the best m_{try}.

In summary, random forests are a substantial modification of bagging that build a large collection of **de-correlated** trees, and then average them. A random forest is an accurate predictor, relatively robust to outliers and noise, faster than bagging and boosting, simply parallelized and it is easy to tune the parameters. Apart from these, random forests can estimate variable importance, which will be touched again in chapter 5.

5.1.3 Boosting

Boosting is a general method which attempts to "boost" the accuracy of any given learning algorithm. It is motivated by the question from Kearns and Valiant [76]:"Can a set of weak learners create a powerful committee?" A weak learner is regarded as a classifier which performs just slightly better than random guessing. In contrast, a powerful committee is a classifier that is well-correlated with the true classification. One of the earliest boosting algorithms is the hypothesis boosting mechanism proposed by Schapire in [72], but it was not adaptive and could not take full advantage of the weak learners. In 1996, Schapire presented AdaBoost in [77]. It can adapt to weak learners, and more important it was a breakthrough which showed that a strong learner can be produced in polynomial time. After that, various boosting algorithms were proposed. In our thesis, we mainly concentrate on the AdaBoost algorithm, sometimes called AdaBoost.M1.

Like Bagging, the AdaBoost algorithm generates a set of classifiers and lets them vote. Beyond this, the two algorithms differ substantially. AdaBoost, shown in algorithm 3, generates the classifiers sequentially, while Bagging can generate them in parallel. AdaBoost also changes the weights of the training instances provided as input to each inducer based on classifiers that were previously built. The goal is to force the inducer to minimize the expected error over different input distributions. Given an integer M specifying the number of trials, M weighted training sets S_1, S_2, $\cdots$, S_M are generated in sequence and T classifiers $C1, C2, \cdots, C_M$ are built. A final classifier C^* is formed using a weighted voting scheme: the weight of each classifier depends on its performance on the training set used to build it.

Algorithm 3 AdaBoost. M1.

Input: training set S of size N, Inducer I, integer M (number of trials)

1. Initialize the observation weights $w_i = 1/N$, $i = 1, 2, \cdots, N$.
2. For $m = 1$ to M
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, 2, \cdots, N$.
   (e) Normalize w_i, $i = 1, 2, \cdots, N$.
3. $C^*(x) = \text{sign}[\sum_{m=1}^{M} \alpha_m G_m(x)]$

Output: classifier C^*

Freund [73] proved that the ensemble (training) error E is bounded from above

$$E < 2^M \prod_{m=1}^{M} \sqrt{\text{err}_m(1 - \text{err}_m)}.$$

Since the err_m of a weak learner is strictly less than 0.5, the ensemble error E monotonically decreases as new classifiers are added.

Once the training is complete, test data are classified by this ensemble of T classifiers using weighted majority voting, where each classifier receives a voting weight that is inversely proportional to its normalized error. The weighted majority voting then chooses the class receiving the highest total vote from all classifiers.

5.1.4 Experiments and discussion

After introducing these ensemble methods, we set up two groups of experiments:

- comparing the basic classifiers and bagging of basic classifiers;
- comparing performance of ensemble methods.

Table 5.2 assembles all testing results. Similar to the previous chapter, useful information is extracted to form some new tables or figures.

Bagging with basic classifiers 50 bootstrap samples bagging is tested here. Figure 5.1 plots the bar charts of the AUC of basic classifiers and bagging. It illustrates that all results of bagging (right bars) exceed the performance of the basic classifiers (left bars). If we use a student t-test here to compare the difference of AUC with and without bagging, the p-values are 0.4592 for case-based reasoning, 0.10368 for Naive Bayes, 0 for the Decision tree and 0.31978 for logistic regression. Although bagging helps all four basic classifiers, applying it to the Decision tree gives the most significant difference. The results fit the theoretical analysis in subsection 5.1.1 that the Decision tree, which is a kind of unstable classifier, can get more benefit from bagging.

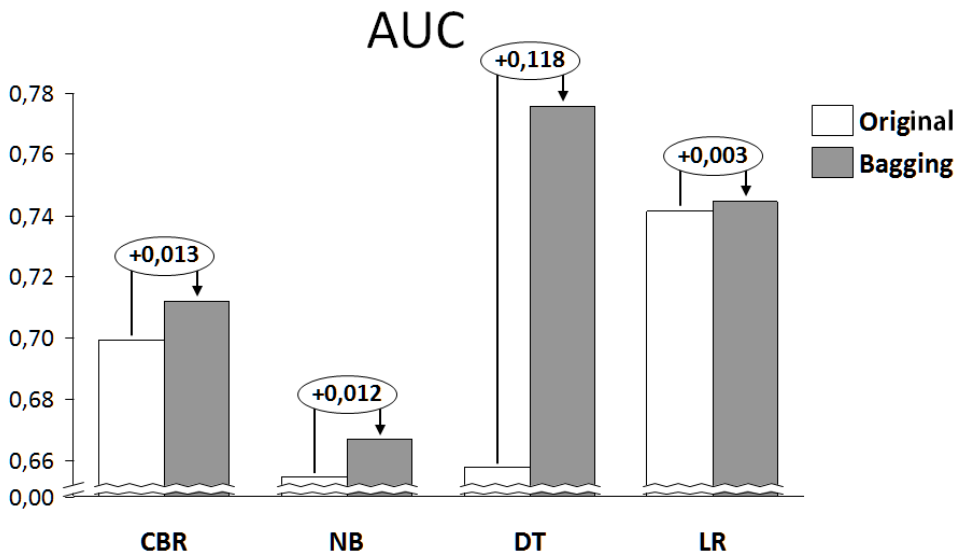


Figure 5.1: Bar charts of the AUC value with and without bagging.

Comparing ensemble methods We configure random forests and AdaBoost as following:

- Random forests: 50 trees are adopted and m_{try} is set to the defaulted value $\sqrt{p} = 10$.
- AdaBoost: 50 boosting iterations. Decision stump, also called 1-rules [78], is used in conjunction with AdaBoost. It is a one level decision tree, which gives a classification based on only a single input feature, independent of being a nominal feature or numeric feature. Weights of instances are updated by each AdaBoost iteration (w_i in algorithm 3) and can be input into the decision stump easily to make the best split for the next AdaBoost iterations.

By observing the singular metric (TP rate, precision, TN rate, etc.) results in table 5.2, both random forests and AdaBoost are highly biased to the majority class, in particular for AdaBoost, since all new arrear customers are classified as delayers when we use 0.5 as the cutoff threshold to distinguish defaulters and delayers. It thus proves again that just taking singular metrics into account is not a comprehensive way to measure the performance. On the other hand, random forests, AdaBoost and bagging with decision tree generate the highest AUC so far, which throw off the basic classifiers remarkably.

Until now, we have introduced ensemble methods and experimentally studied their usage in this problem. Compared with the original classifiers, the ensemble methods show great power to lift the performance. However, it is noted that undersampling with logistic regression is even better than bagging with logistic regression. So, we will combine sampling techniques with ensemble methods in the next section, and try to obtain benefits from both techniques.

5.2 Ensemble method application in imbalanced learning

In recent years, ensembles of classifiers have arisen as a possible solution to the class imbalance problem attracting great interest among researchers. Galar et al. reviewed the existing works in [79], and categorize them into four different families as shown in figure 5.2. In this section, we will focus on some methods which will be employed in our thesis, i.e., symmetric bagging, balanced random forests, EasyEnsemble and balanceCascade.

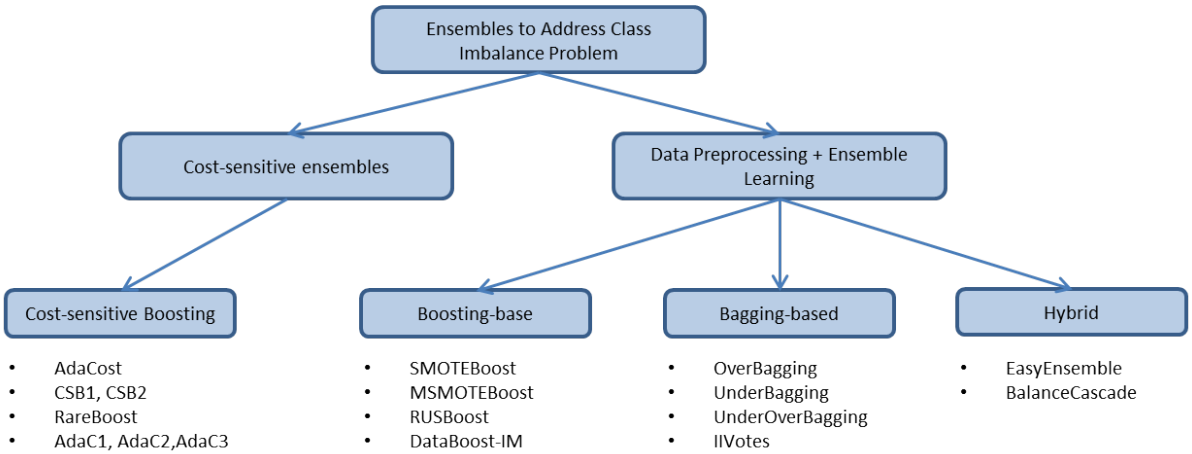


Figure 5.2: Proposed taxonomy for ensembles to address the class imbalance problem [79].

5.2.1 Symmetric bagging

Although the testing result in subsection 5.1.4 has already shown that bagging trees outperforms a single decision tree, Bagging is still unavoidably awkward when combined with imbalanced data: during the bootstrap step, there is a significant probability that a bootstrap sample contains even fewer instances of the minority class. A symmetric bagging strategy, as shown in algorithm 4, was proposed by Hido et al. in [80] to overcome it. On the one hand, undersampling is an efficient strategy to deal with class imbalance as shown in the previous chapter. On the other hand, multiple bootstrap steps can suppress the drawback brought by undersampling that throws away a lot of potentially useful data.

5.2.2 Balanced random forests

Random forests are a fancier version of bagging decision trees. A decision tree is constructed to minimize the overall error rate. When combined with imbalanced data, it will tend to bias the majority class, which often results in poor accuracy for the minority class. In addition, as mentioned in the previous subsection, fewer instance of the minority class in a bootstrap sample also disturbs the performance.

Algorithm 4 Symmetric Bagging

Input: training set S, Inducer I, integer M (number of bootstrap samples)
for $i = 1$ to M **do**
  Draw a bootstrap sample from the minority class.
  Randomly draw the same number of cases with replacement from the majority class.
  $S' =$ bootstrap sample from S (i.i.d. sample with replacement)
  $C_i = I(S')$
end for
$C^*(x) = \arg\max_{y \in Y} \sum_{i:C_i(x)=y} 1$ (the most often predicted label y)
Output: classifier C^*

Chen et al. proposed a solution called balanced random forests (BRF) in [81] as shown in algorithm 5. Unlike random forests, balanced downsampled data are used to build trees. It is easy to see that balanced random forests is computationally more efficient with very imbalanced data, since each tree only uses a small portion of the training set to grow.

Algorithm 5 Balanced Random Forests

1. For $b = 1$ to M:
   (a) Draw a bootstrap sample from the minority class.
   (b) Randomly draw the same number of cases with replacement from the majority class.
   (c) Grow a random-forest tree T_b using the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^M$.

5.2.3 EasyEnsemble

As aforementioned, bagging mainly reduces the variance, while boosting mainly reduces the bias. Borrowing this idea, Liu et al. proposed EasyEnsemble in [61]. Algorithm 6 shows how EasyEnsemble works. It generates T balanced subproblems by bootstrapping and undersampling. The output of the ith subproblem is AdaBoost classifier H_i, an ensemble with s_i weak classifiers $\{h_{i,j}\}$. EasyEnsemble uses Bagging as the main ensemble learning method, but in spite of training a classifier for each new bag, they train each classifier using AdaBoost. Hence, the final classifier is an ensemble of ensembles and can reduce the bias and variance simultaneously. The same as other bagging based classifiers, EasyEnsemble can be trained in parallel.

5.2.4 BalanceCascade

EasyEnsemble is an unsupervised strategy to explore the dataset since it uses independent random sampling with replacement. BalanceCascade, another algorithm proposed in [61], explores the dataset in a supervised manner. The idea is as follows. After H_1 is trained, if an example $x_1 \in N$ is correctly classified to be in the majority class by H_1, it is reasonable to conjecture that x_1 is somewhat redundant in N, given that we already have H_1. Thus, we can remove some correctly classified majority class examples from N. As in EasyEnsemble, AdaBoost is employed in this method. The pseudocode of BalanceCascade is shown in Algorithm 7.

Algorithm 6 EasyEnsemble.

Input: A set of minority class examples P, a set of majority class examples N, $|P| < |N|$, the number of subsets M to sample from N, and s_i, the number of iterations to train an AdaBoost ensemble H_i.

1. For $i = 1$ to M
   (a) Randomly sample a subset N_i from N, $|N_i| = |P|$.
   (b) Learn H_i using P and N_i. H_i is an AdaBoost ensemble with s_i weak classifiers $h_{i,j}$ and corresponding weights $\alpha_{i,j}$. The ensemble's threshold is θ_i, i.e.,

$$H_i(x) = \text{sgn}\left(\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \theta_i\right)$$

2. $C^*(x) = \text{sgn}\left(\sum_{i=1}^{M} \sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \sum_{i=1}^{M} \theta_i\right)$

Output: classifier C^*

Algorithm 7 BalanceCascade.

Input: A set of minority class examples P, a set of majority class examples N, $|P| < |N|$, the number of subsets M to sample from N, and s_i, the number of iterations to train an AdaBoost ensemble H_i.

1. $f = \sqrt[M-1]{|P|/|N|}$, f is the false positive rate (the error rate of misclassifying a majority class example) that H_i should achieve.
2. For $i = 1$ to M
   (a) Randomly sample a subset N_i from N, $|N_i| = |P|$.
   (b) Learn H_i using P and N_i. H_i is an AdaBoost ensemble with s_i weak classifiers $h_{i,j}$ and corresponding weights $\alpha_{i,j}$. The ensemble's threshold is θ_i, i.e.,

$$H_i(x) = \text{sgn}\left(\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \theta_i\right)$$

   (c) Adjust θ_i, such that H_i's false positive rate is f.
   (d) Remove from N all examples that are correctly classified by H_i.
3. $C^*(x) = \text{sgn}\left(\sum_{i=1}^{M} \sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \sum_{i=1}^{M} \theta_i\right)$

Output: classifier C^*

5.2.5 Experiments and discussion

In this subsection, we will first compare and discuss the impact of sampling techniques on ensemble methods and the performance of symmetric bagging, balanced random forests, EasyEnsemble and BalanceCascade and find the best approach so far. All results are summarized in table 5.3. We will extract some data and form new tables and figures in order to discuss conveniently. The methods used in this subsection are configured as below:

- Symmetric bagging: 50 bootstrap samples, 1:1 sampling and 1:5 sampling;
- Balanced random forests: 50 trees are adopted and the number of features (m_{try}) is set to the default value $\sqrt{p} = 10$; 1:1 bootstrap sampling;
- EasyEnsemble: 50 bootstrap samples for bagging, AdaBoost uses 20 boosting iterations, decision stump as weak classifier;
- BalanceCascade: the same configuration as EasyEnsemble.

Ensemble methods with sampling techniques We compare the impact of sampling techniques on ensemble methods experimentally. Bagging with decision trees, bagging with logistic regression and random forests are tested with three kinds of sampling ratios, 1:9 (original), 1:5 (around half undersampling delayers) and 1:1 (balanced sampling). Figure 5.3 illustrates the different AUC values and all three groups of results show the same trend that balanced sampling does help the classification. The original distribution (the left most bars) obtains the lowest AUC value, 1:5 ratio (the bars in the middle) lifts the AUC value, and 1:1 symmetric sampling (the rightmost bars) gives the highest AUC scores. Our testing results are consistent with former studies of symmetric bagging [80] and balanced random forests [81].

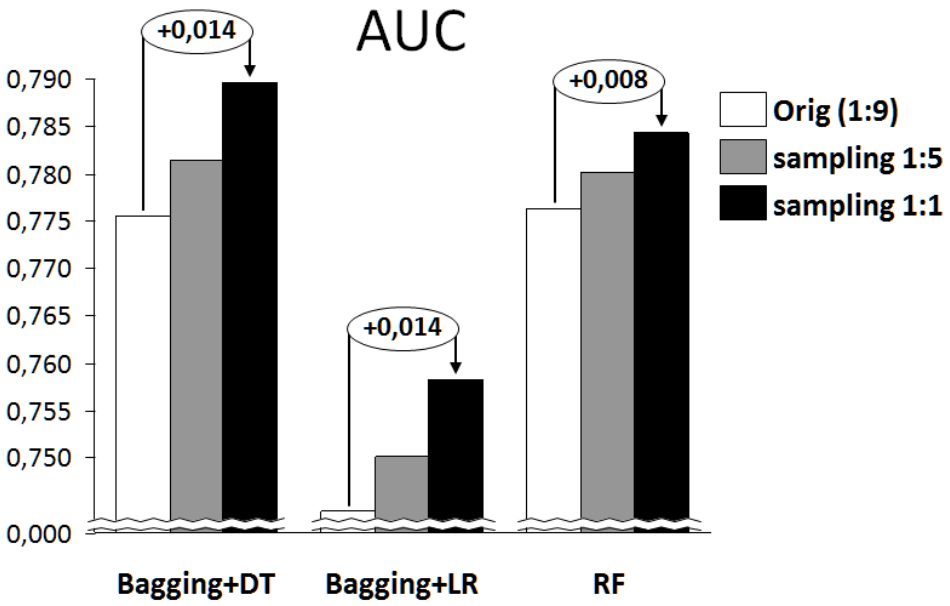
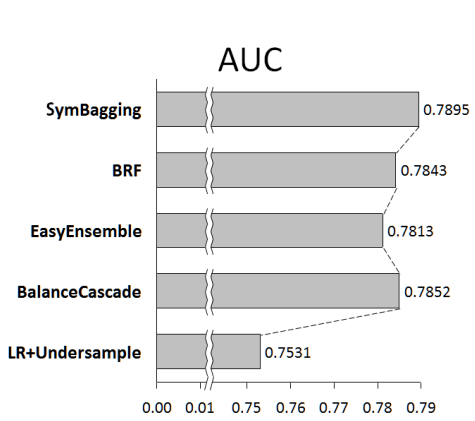


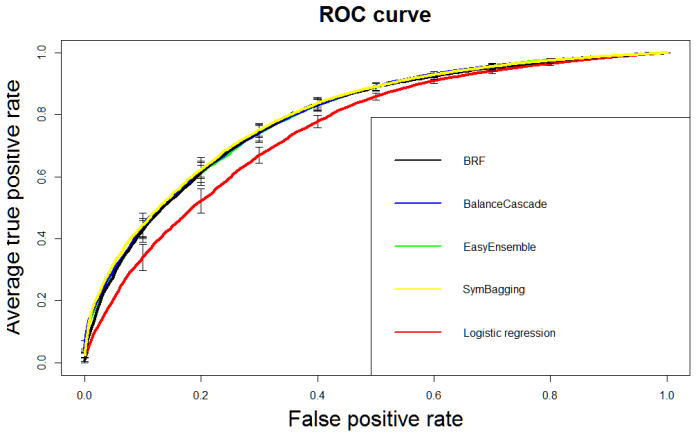
Figure 5.3: Bar chart of AUC values with different sampling ratios. Bagging+DT, bagging+LR and RF are short for bagging with decision tree, bagging with logistic regression and random forests, respectively.

Balanced ensemble methods Symmetric bagging, balanced random forests, EasyEnsemble and BalanceCascade are compared in this experiment. Figure 5.4a plots the bar charts of AUC values. The performance of all four classifiers are close to each other. Although symmetric bagging performs a bit better than the other three methods, the student t-test does not show significant differences between them (p-value between symmetric bagging and balanced random forests is 0.6415354). Figure 5.4b illustrates the ROC curves. The curves of four balanced ensemble methods are very close and even overlapping with each other. The curve of symmetric bagging is slightly more to the most left upper point than other curves on some parts of the curve (like in the left corner), but it cannot dominate the whole ROC graph.

We also paste the result of logistic regression with undersampling in the figures as baseline. Apparently its performance is exceeded by all four balanced ensemble methods.



(a) Bar chart of AUC values.



(b) ROC curves. The bars attached on the curves are standard deviation bars.

Figure 5.4: Testing results of symmetric bagging, balanced random forests, EasyEnsemble and BalaceCascade. SymBagging, BRF and LR are abbreviations of symmetric bagging, balanced random forests and logistic regression.

In summary, these experiments tell us two things: balancing the number of defaulters and delayers conjuncted with ensemble methods does help the classification; balanced random forests, BalanceCascade and symmetric bagging achieve the best performance so far.

Since we have already tested various classifiers, why not integrate their advantage and obtain a better classifier? The next section will discuss combining different classifiers.

5.3 Combining multiple learners

Learning a classifier is an ill-posed problem because the data employed as training set is always a finite sample of real circumstances. Each algorithm must come with a set of assumptions, i.e., the data distribution, noise model, parameters, etc. One assumption may be suitable for one situation, while failing under different situations. For example, the performance of a learner may be fine-tuned to get the highest possible accuracy on a validation set, but the model after fine-tuning still performs not accurate enough when it encounters new coming testing data. The idea is that there may be another base-learner that is accurate on these. By suitably combining multiple base learners, the accuracy can be improved. This section will introduce the two most popular and basic combining strategies, voting and stacking, and we use them to construct a better classifier.

5.3.1 Voting

Voting is the simplest way to combine multiple classifiers. For classification problems, majority voting gives the final classification result; for regression problems, voting corresponds to a linear combination of the learners (see figure 5.5a):

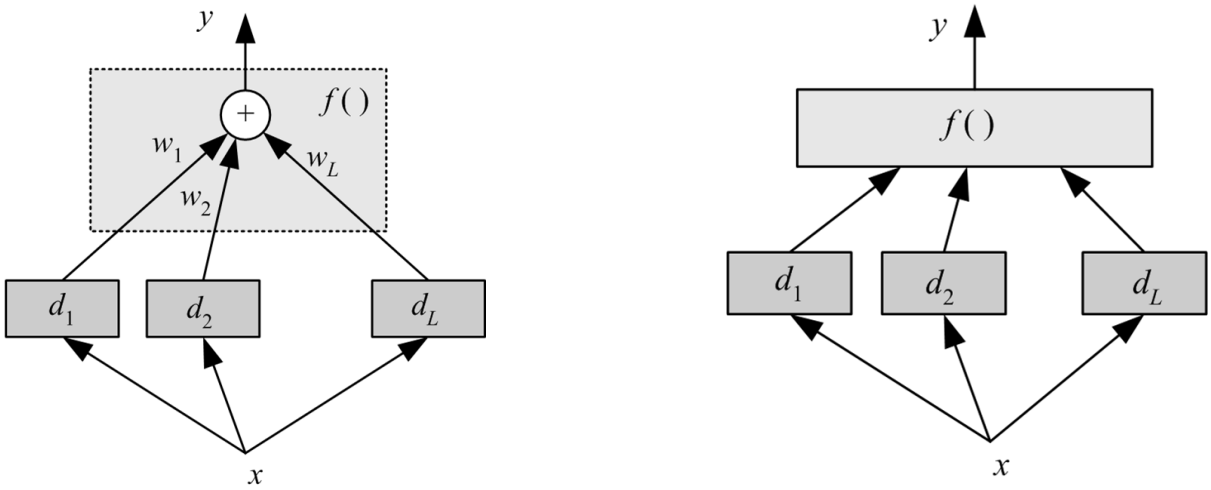
$$y_i = \sum_j w_j d_{ji}, \text{where } w_j \geq 0, \sum_j w_j = 1$$

In the simplest case, all learners are given equal weight. We will use the average class probability on the experiment. Actually, voting is a special algebraic combiner. There are also some other combiners such as maximum rule, minimum rule, median rule, product rule. As we have already seen in the introduction of bagging, voting can reduce the variance of different base classifier.

5.3.2 Stacking

As shown in figure 5.5b, stacking is concerned with combining multiple classifiers generated by using different learning algorithms $L_1, \cdots, L_N$ on a single dataset S. In the first phase, a set of base-level classifiers $d_1, d_2, \cdots, d_N$ is generated, where $d_i = L_i(S)$. The outputs of the base-learners d_j define a new L dimensional space. In the second phase, a meta-level classifier is learned from the new L dimensional data.

In stacking, there is no restriction on the combiner function and unlike voting, $f(\cdot)$ can be any arbitrary combiner method, such as regression splines or an ensemble method. But in practice, a logistic regression model is often used as the combiner.



(a) Voting

(b) Stacking

Figure 5.5: Combining multiple classifiers [82].

5.3.3 Experiments and discussion

We use the equal weight average in voting, and logistic regression as the meta classifier in stacking. 10-fold cross validation is employed. The best classifiers we got so far are assembled in the experiments. These "basic" classifiers are:

- Symmetric bagging: 50 bootstrap samples and 1:1 sampling
- Balanced random forests: 50 trees are adopted and the number of features (m_{try}) uses the default value $\sqrt{p} = 10$; 1:1 bootstrap sampling;
- EasyEnsemble: bagging with 50 bootstrap samples and AdaBoost with 20 boosting iterations; decision stump is used as weak classifier in AdaBoost;
- BalanceCascade: the same configuration as EasyEnsemble.

Table 5.4 records the testing result. In order to observe easily, we put AUC values of voting, stacking and "basic" classifiers together in table 5.1. Although voting and stacking is slightly better than the four "basic" methods, it is not significantly better, e.g., p-value is 0.5911 between stacking and balanced random forests.

Methods	SymmBagging	BRF	EE	BC	Voting	Stacking
AUC	0.7895	0.7843	0.7813	0.7852	0.7898	0.7916

Table 5.1: AUC values of "basic" classifiers, voting and stacking.

5.4 Summary

In this chapter, we first introduced ensemble methods and compared them with basic classifiers, then we comparatively studied the impact of ensemble methods with sampling techniques on the imbalanced learning. We can summarize that:

- Although bagging is one of the most intuitive and simplest ensemble methods, it has a surprisingly good performance. Bagging is a relatively easy way to improve an existing method.
- Bagging, random forests and AdaBoost outperform the basic classifiers, but they are still biased to the majority class.
- Sampling methods help improving the performance of ensemble methods. The more balance between the majority class and minority class, the better the performance is.
- Symmetric bagging with decision trees, BalanceEnsemble and balanced random forests give the best results so far.
- Although the average values of the cross validation results of stacking and voting are slightly better than the results of the balanced ensemble methods, they do not perform significantly better.

Methods	Recall	Precision	FN rate	F-measure	G-means	AUC
CBR	0.0576 ± 0.110	0.3884 ± 0.066	0.9962 ± 0.012	0.1005 ± 0.031	0.2377 ± 0.051	0.6989 ± 0.018
Bagging + CBR	0.0342 ± 0.083	0.4000 ± 0.058	0.9965 ± 0.011	0.1145 ± 0.035	0.1850 ± 0.024	0.7017 ± 0.083
NB	0.2092 ± 0.029	0.3964 ± 0.002	0.9671 ± 0.038	0.3966 ± 0.004	0.4498 ± 0.030	0.6540 ± 0.012
Bagging + NB	0.1931 ± 0.016	0.4012 ± 0.005	0.9683 ± 0.018	0.3974 ± 0.031	0.4324 ± 0.022	0.6664 ± 0.026
DT	0.1141 ± 0.017	0.4181 ± 0.062	0.9830 ± 0.004	0.1785 ± 0.024	0.3340 ± 0.025	0.6574 ± 0.018
Bagging + DT	0.0885 ± 0.008	0.5438 ± 0.053	0.9920 ± 0.002	0.1519 ± 0.011	0.2961 ± 0.013	0.7754 ± 0.012
LR	0.0509 ± 0.007	0.4584 ± 0.076	0.9935 ± 0.002	0.0915 ± 0.012	0.2245 ± 0.015	0.7412 ± 0.017
Bagging + LR	0.0536 ± 0.023	0.4506 ± 0.010	0.9935 ± 0.024	0.4515 ± 0.028	0.2308 ± 0.008	0.7442 ± 0.011
Random Forests	0.0560 ± 0.013	0.5126 ± 0.097	0.9942 ± 0.002	0.1006 ± 0.022	0.1684 ± 0.029	0.7763 ± 0.016
AdaBoost + DecisonStump	0.0 ± 0.0	0.0 ± 0.0	1.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.7747 ± 0.013

Table 5.2: Testing results of basic classifiers with and without Bagging (50 bootstrap samples), Random Forests (50 trees) and AdaBoost(50 iterations with decision stump). The recall, precision, TN rate, F-measure and G-means are achieved when the cut-off threshold is **0.5**, which is the default threshold of these classification methods. As discussed in the previous chapters, these classifiers are biased to the majority classes seriously, especially the AdaBoost algorithm, for which all the customers are classified as delayers!

Methods	Recall	Precision	TN rate	F-measure	G-means	AUC
Bagging + DT	0.0885 ± 0.008	0.5438 ± 0.053	0.9920 ± 0.002	0.1519 ± 0.011	0.2961 ± 0.013	0.7754 ± 0.012
Bagging (1:5) + DT	0.2509 ± 0.033	0.4587 ± 0.016	0.9691 ± 0.005	0.3236 ± 0.041	0.4931 ± 0.021	0.7813 ± 0.028
Symmetric Bagging (1:1) + DT	0.6348 ± 0.029	0.2589 ± 0.009	0.8093 ± 0.007	0.3677 ± 0.013	0.7166 ± 0.016	0.7895 ± 0.024
Bagging + LR	0.0536 ± 0.023	0.4506 ± 0.010	0.9935 ± 0.024	0.4515 ± 0.028	0.2308 ± 0.008	0.7442 ± 0.011
Bagging (1:5) + LR	0.1303 ± 0.009	0.4154 ± 0.008	0.9815 ± 0.016	0.1982 ± 0.037	0.3574 ± 0.057	0.7500 ± 0.016
Symmetric LR (1:1) + LR	0.6054 ± 0.017	0.2283 ± 0.022	0.7859 ± 0.16	0.3325 ± 0.009	0.6891 ± 0.027	0.7581 ± 0.020
Random Forests	0.0560 ± 0.013	0.5126 ± 0.097	0.9942 ± 0.002	0.1006 ± 0.022	0.1684 ± 0.029	0.7763 ± 0.016
Random Forests (1:5)	0.1609 ± 0.021	0.4659 ± 0.063	0.9805 ± 0.003	0.2387 ± 0.029	0.2734 ± 0.033	0.7801 ± 0.010
Balance random forests	0.5563 ± 0.022	0.2985 ± 0.016	0.8628 ± 0.007	0.3884 ± 0.016	0.4073 ± 0.016	0.7843 ± 0.009
EasyEnsemble	0.5836 ± 0.029	0.2796 ± 0.012	0.8422 ± 0.006	0.3780 ± 0.016	0.7009 ± 0.018	0.7813 ± 0.032
BalanceCascade	0.2710 ± 0.025	0.4385 ± 0.052	0.9632 ± 0.006	0.3330 ± 0.029	0.3442 ± 0.031	0.7852 ± 0.013

Table 5.3: Testing results of Symmetric bagging (50 bootstrap samples), balanced random forests (50 trees), EasyEnsemble (bagging with 50 bootstrap samples, AdaBoost with 20 iterations), BalanceCascade (bagging with 50 bootstrap samples, AdaBoost with 20 iterations). The recall, precision, TN rate, F-measure and G-means are achieved when the cut-off threshold is 0.5, which is the default threshold of these classification methods. The results show that the balanced sampling techniques not only suppress the bias for delayers, but also increase the AUC values, which represent the holistic performance.

Methods	Recall	Precision	TN rate	F-measure	G-means	AUC
Voting	0.3258 ± 0.032	0.4302 ± 0.012	0.9556 ± 0.009	0.4303 ± 0.021	0.5572 ± 0.017	0.7898 ± 0.011
Stacking	0.4745 ± 0.014	0.3386 ± 0.037	0.9027 ± 0.032	0.3388 ± 0.010	0.6538 ± 0.036	0.7916 ± 0.038

Table 5.4: Testing results of voting and stacking. The basic classifiers are symmetric bagging, balance random forests, EasyEnsemble and BalanceCascade, which configurations are the same as in table 5.3. Voting uses the average probability, and stacking uses logistic regression as meta classifier. The recall, precision, TN rate, F-measure and G-means are achieved when the cut-off threshold is 0.5, which is the default threshold of these classification methods.

Chapter 6

Knowledge discovery

6.1 Finding the best model

We have already tested and compared several approaches. The experiments in the previous chapter conclude that balanced ensemble methods give the best results. For the sake of building a robust, meaningful and fast model for the Arrears department in ING, we select balanced random forests (BRF). The reasons are as following:

- BRF can handle thousands of variables efficiently. Normally, data preprocessing and feature selection take considerable effort in the pipeline of the classification system, especially on high dimensional data. However, RF needs less data preprocessing (it can handle both discrete and continuous data), is not sensitive to outliers, and does not need variable deletion [70]. Plus the excellent performance, random forests is known as an "off-the-shelf" machine learning algorithm.
- BRF is a fast method. Our experiment environment is a server with four 2.67 GHz Intel CPUs, 16 GB RAM and Windows 2008 Server operating system. BRF is the fastest methods in all tested balanced ensemble methods and figure 6.1 demonstrates it. There are three reasons which make BRF fast:
  - BRF undersamples the majority class in each bootstrap sample, which reduces the training data size;
  - Random forests only uses a part of the features (m_{try}) but not the full set when it constructs each split node;
  - Random forests does not need to prune trees.
  - Random forests are extremely parallel because bootstrapping is easy to parallelize. In contrast, it can be significantly harder to parallelize a boosting-based method.
- BRF does not need too many parameters to tune. The only parameters are the number of trees (N_{trees}) and the number of variables randomly sampled as candidates at each split (m_{try})

6.1.1 Tuning parameters

In order to find the best m_{try} and N_{trees}, we set dozens of runs to iterate m_{try} from 1 to 50 and N_{trees} from 50 to 2000. 10-fold cross validation is employed in each run. Table 6.9 lists the testing results with mean and standard deviation. The bar graphs 6.2 are drawn from the data extracted from table 6.9 and we can get the conclusions as below:

- There is no overfitting when more trees are added: the AUC value keeps increasing along with adding more trees in figure 6.2a. $N_{tree} = 2000$ can always beat less trees. However, it is a remarkable fact that the benefit from increasing N_{tree} becomes slower because the growth is gradually flat with large N_{tree}.

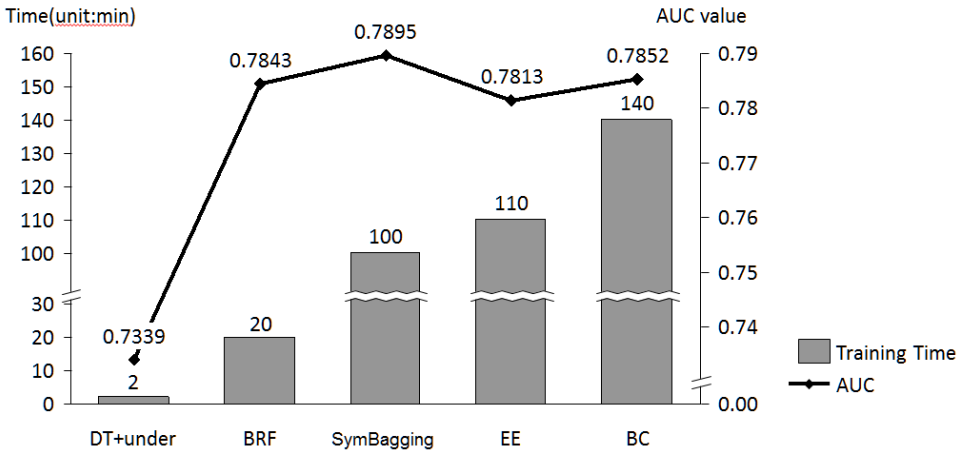


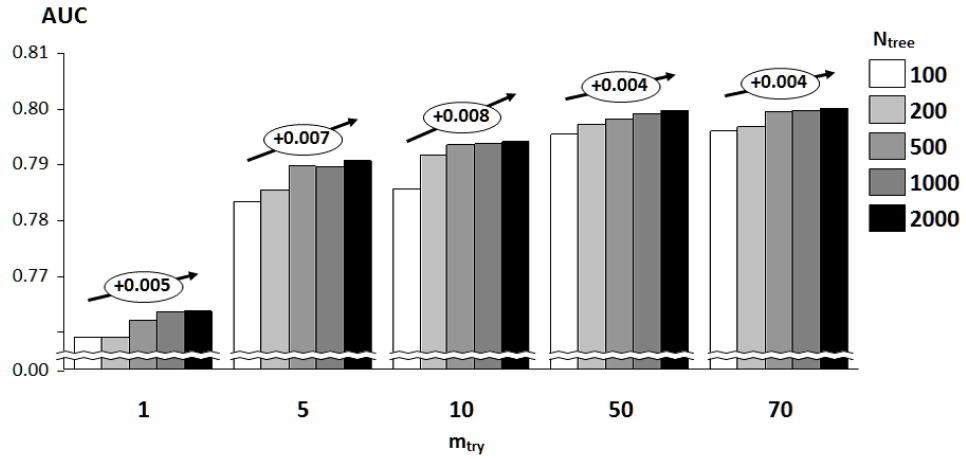
Figure 6.1: The comparison of the training time of classifiers. The bars use the left Y-axis and show the approximate running time for one round of 10-fold cross validation. The line chart uses the right Y-axis and lists the performance of each classifier. The single decision tree with undersampling (the leftmost bar) is also pasted here as baseline. Clearly, in the four balanced ensemble methods which output similar AUC values, balanced random forests costs the least time. DT+under, BRF, SymBagging, EE and BC are abbreviation of decision tree with undersampling, balanced random forests, symmetric bagging, EasyEnsemble and BalanceCascade, respectively. The results of voting and stacking are not included here because their time consumptions are the sum of four basic classifiers and the performances are not significantly better.

- The performance varies with the change of m_{try}. Basically, accompanied with increasing m_{try}, the AUC value becomes higher. However, the performance drops when m_{try} reaches a certain value. In figure 6.2b, the bar of $m_{try} = 80$ (the right most) is shorter than 70 (the second right most). It is noteworthy that the default value of m_{try}, which is 10 in our case, does not work as well as the larger m_{try}, and it implies that tuning random forests is an essential step.

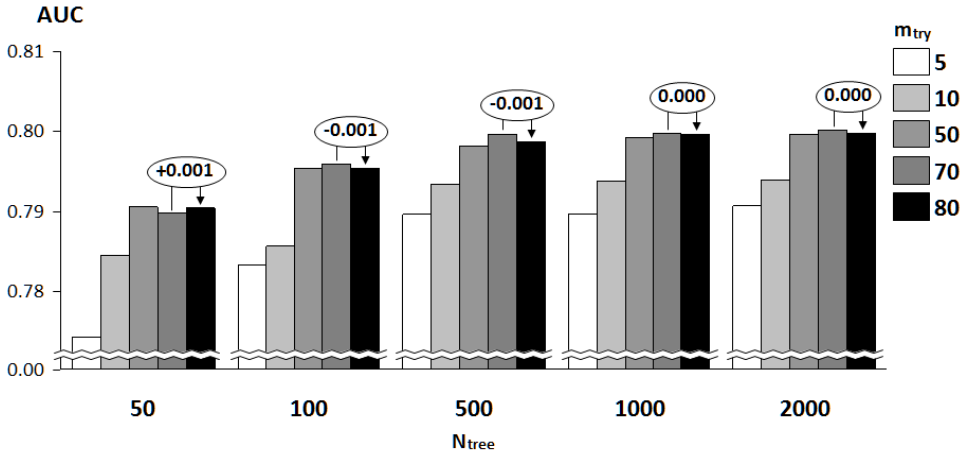
6.1.2 Building the best model

According to the experiments we did, we choose balanced random forests with $\mathbf{N_{tree} = 2000}$ and $\mathbf{m_{try} = 70}$ to build our final model. Unlike randomly dividing the training set and the testing set in the cross validation as former experiments, it is more natural and reasonable to use all new customers from one or more months as the testing set, and all other customers as the training set. The reason is that the model always predicts new arrear customers from a new coming month when it is deployed in the Arrears department. The latent factor of a new coming month may be different from old months, like the economic situation, different consumption expectation (e.g., more consumption in holidays), etc. So, this kind of experiment setting conforms the real application scenarios. However, it can be foreseen that the testing result will be worse than the results in table 6.9, because the former testing samples 10-fold sets randomly so that some instances of the testing set and the training set are from the same months, but now the testing set is strictly isolated from the training set.

The whole dataset spans from November 2011 to March 2013. We employ the first 12 months (November 2011 to October 2012) as training set, and the remaining 5 months to estimate the performance of the model. Table 6.1 demonstrates the result and the average AUC is **0.78565**. It confirms our guess that the result is worse than the results of cross validation. It also shows that the results vary in different months. The test sets of November and December 2012 have better results than the test sets of the months in 2013. But the AUC values have no obvious linear relation with the percentage of defaulters or the number of new arrear customers. One possible explanation is that the test set of 2012 is closer to the training set, so that the latent factors are more similar. But another group of tests in table 6.2 disproves this. So, we can only conjecture that the difficulty of predicting varies in different months.



(a) Bar chart of AUC values. The clusters of columns are when m_{try} equals 1, 5, 10, 50 and 70. Each cluster contains five columns from white to black which N_{tree} are 100, 200, 500, 1000 and 2000 respectively. The AUC values grow with the increasing of N_{tree}.



(b) Bar chart of AUC values. The clusters of columns are when N_{tree} equals 100, 200, 500, 1000 and 2000. Each cluster contains five columns from white to black when m_{try} is 5, 10, 50, 70 and 80 respectively. The AUC values increase while enlarging m_{try}. However, AUC values start falling down after m_{try} is large than 70.

Figure 6.2: Tune m_{try} and N_{tree} of random forests.

Month	Nov 2012	Dec 2012	Jan 2013	Feb 2013	Mar 2013
# New arrear customers	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
# Defaulters	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
# Delayers	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
percentage of defaulters	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
AUC	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

Table 6.1: The AUC values of different months. We use the data from [redacted] to train the model, and test the new arrear customers from [redacted].

Training months \ Test month	Nov 2012	Dec 2012	Jan 2013	Feb 2013	Mar 2013
Nov 2011 to Oct 2012	0.7959	0.8014	0.7796	0.7740	0.7774
Dec 2011 to Nov 2012		0.8004	0.7815	0.7601	0.7763
Jan 2012 to Dec 2012			0.7797	0.7754	0.7723
Feb 2012 to Jan 2013				0.7755	0.7733
Mar 2012 to Feb 2013					0.7728

Table 6.2: The AUC values of five tests. In each test, the training set contains 12-months of new arrear customers. If our assumption is true that the result should be better if the training set is closer to the test set, the AUC value of each column should increase. But this is not the case, so our assumption does not hold.

6.2 Cost analysis

Both from the customer and bank perspective, the objective is to save as much on risk costs as possible. In addition to this, the bank needs to balance these risk costs with operational costs, such as employing customer service clerks and system capabilities. In this section, we first propose a cost matrix, then analyse our project in two ways, i.e., making optimal decisions by Bayesian Decision Theory, and deciding the best cut-off threshold to make minimal global costs.

6.2.1 Cost matrix

In our case, contacting new arrear customers after classification will import operational costs. As mentioned in chapter 1, only letter/Email/SMS will be sent to the new arrear customers who are predicted as delayers, and more intensive contacting such as phone calls will be given to the new arrear customers who are predicted as defaulters. All these treatments will cost ING human resources and overhead.

We also define the processing benefit as the risk cost reduction brought by the treatment. Since a defaulter will repay the debt after the treatment with a certain probability, the interest loss, provision (explained in figure 1.2b) and potential collection loss will be saved.

The operational cost matrix and processing benefit matrix are listed as in table 6.3a and 6.3b. The final cost matrix, which is shown in table 6.3c, is the operational cost matrix subtracting the processing benefit matrix. **Appendix A will explain the interest loss, the provision, the potential collection loss, and how we get the value of these losses.**

		Predict class	
		Defaulter	Delayer
Actual class	Defaulter		
	Delayer		

(a) Operational cost matrix.

		Predict class	
		Defaulter	Delayer
Actual class	Defaulter		
	Delayer		

(b) Processing benefit matrix.

		Predict class	
		Defaulter	Delayer
Actual class	Defaulter		
	Delayer		

(c) Cost matrix.

Table 6.3: Operational cost matrix, processing benefit matrix and cost matrix. Units of the numbers in the matrices are Euro. Appendix A will explain how we get these numbers.

6.2.2 Minimum-error-rate classification

For the sake of expressing easier in the formula, let us denote r as defaulter and g as delayer (in the process, defaulters are called "red" customers and delayers are "green"). Consequently, ω_r represents the actual label defaulter, and ω_g is the actual label delayer. Given a new arrear customer x, the probabilities of x being defaulter and delayer are $P(\omega_r|x)$ and $P(\omega_g|x)$, respectively. We denote α as our action/decision, i.e., α_r means we predict the new arrear customer as defaulter, and α_g as delayer. By definition we can also incur the cost $\lambda(\alpha_i|\omega_j)$, where $i, j \in \{r, g\}$. Table 6.3c can also be rewritten as table 6.4:

		Predict class	
		Defaulter	Delayer
Actual class	Defaulter	$\lambda(\alpha_r\|\omega_r) =$ [redacted]	$\lambda(\alpha_g\|\omega_r) =$ [redacted]
	Delayer	$\lambda(\alpha_r\|\omega_g) =$ [redacted]	$\lambda(\alpha_g\|\omega_g) =$ [redacted]

Table 6.4: Cost matrix.

The expected loss associated with taking action α is merely:

$$L(\alpha_r|x) = \lambda(\alpha_r|\omega_r)P(\omega_r|x) + \lambda(\alpha_r|\omega_g)P(\omega_g|x) \tag{6.1}$$

$$L(\alpha_g|x) = \lambda(\alpha_g|\omega_r)P(\omega_r|x) + \lambda(\alpha_g|\omega_g)P(\omega_g|x) \tag{6.2}$$

In this two-class case, the optimal prediction is a defaulter if and only if the expected loss $L(\alpha_r|x)$ is less or equal to $L(\alpha_g|x)$, i.e., if and only if

$$\lambda(\alpha_r|\omega_r)P(\omega_r|x) + \lambda(\alpha_r|\omega_g)P(\omega_g|x) \leq \lambda(\alpha_g|\omega_r)P(\omega_r|x) + \lambda(\alpha_g|\omega_g)P(\omega_g|x)$$

which is equal to

$$(\lambda(\alpha_r|\omega_r) - \lambda(\alpha_g|\omega_r))P(\omega_r|x) \leq (\lambda(\alpha_r|\omega_g) - \lambda(\alpha_g|\omega_g))P(\omega_g|x)$$
$$\frac{P(\omega_r|x)}{P(\omega_g|x)} \leq \frac{\lambda(\alpha_g|\omega_g) - \lambda(\alpha_r|\omega_g)}{\lambda(\alpha_r|\omega_r) - \lambda(\alpha_g|\omega_r)}$$

If this inequality is in fact an equality, then predicting either class is optimal. Since it is a two-category classification, $P(\omega_g|x) = 1 - P(\omega_r|x)$. The threshold for making the optimal decision is p^* such that:

$$\frac{p^*}{1-p^*} = \frac{\lambda(\alpha_g|\omega_g) - \lambda(\alpha_r|\omega_g)}{\lambda(\alpha_r|\omega_r) - \lambda(\alpha_g|\omega_r)}$$

Rearranging the equation for p^* leads to the solution

$$p^* = \frac{\lambda(\alpha_g|\omega_g) - \lambda(\alpha_r|\omega_g)}{\lambda(\alpha_r|\omega_r) - \lambda(\alpha_g|\omega_r) + \lambda(\alpha_g|\omega_g) - \lambda(\alpha_r|\omega_g)} \tag{6.3}$$

By using the real costs in table 6.3c, we get $p^* =$ [redacted] $= 0.0273$. For classifiers which generate real probabilities like Naive Bayes, we employ the cost matrix analysis and get the new results in table 6.5. The classification result is biased to defaulters. It is noted that the AUC value is the same because optimal decisions only change the threshold of the classification.

Methods	Recall	Precision	FN rate	F-measure	G-means	AUC
NB	0.2092	0.3964	0.9671	0.3966	0.4498	0.6540
Cost + NB	0.7321	0.1659	0.6124	0.1651	0.6696	0.6540

Table 6.5: The result of Naive Bayes without and with cost matrix $[-353, 3; 13, 3]$. The decision threshold which classifies defaults and delayers is 0.5 without cost matrix and 0.0273 with cost matrix.

6.2.3 Optimize the total cost

Most classifiers do not output real probabilities but pseudo-probabilities or ranking scores. It is not suitable to use Bayesian decision theory anymore for them. Alternatively, we can calculate the minimal global cost to determine the best cut-off threshold.

As mentioned in chapter 3 in curve metrics, different confusion matrices will be generated by a given different cut-off threshold. Let us denote the threshold as θ, and $TP(\theta)$, $FP(\theta)$, $TN(\theta)$ and $FN(\theta)$ as the four elements in the confusion matrix. Since the elements in the cost matrix represent the average unit cost per customer, we can dot multiply the cost matrix and confusion matrix, and sum up the four elements as total cost.

		Predict class	
		Defaulter	Delayer
Actual class	Defaulter	[redacted] $\cdot TP(\theta)$	[redacted] $\cdot FN(\theta)$
	Delayer	[redacted] $\cdot FP(\theta)$	[redacted] $\cdot TN(\theta)$

Table 6.6: Total cost by given a threshold θ. $C_{\text{total}}(\theta) =$ [redacted] $\cdot TP(\theta) +$ [redacted] $\cdot FP(\theta) +$ [redacted] $FN(\theta) +$ [redacted] $\cdot TN(\theta)$.

By using the same way of plotting a ROC curve, different costs can be calculated by traversing each threshold and plotting the result on a cost curve. Then, the minimal cost can be spotted and the corresponding threshold is just the optimal threshold. We will gives an example. In our test set, there are [redacted] new arrear customers, which contain [redacted] defaulters and [redacted] delayers. Balanced random forests with 2,000 trees and default m_{try} gives the confusion matrix with a threshold $\theta = 0.5$ in table 6.7a, where the total cost is [redacted]. Table 6.7b gives the confusion matrix with a threshold $\theta = 0.178$ where the total cost is minimal, i.e., [redacted]. The confusion matrix with the minimal cost is more biased to the defaulters since the benefit brought by correctly classifying a defaulter is much higher than the cost brought by misclassifying a delayer.

		Predict class	
		Defaulter	Delayer
Actual class	Defaulter	[redacted]	[redacted]
	Delayer	[redacted]	[redacted]

(a) Confusion matrix with threshold 0.5. Total cost is [redacted].

		Predict class	
		Defaulter	Delayer
Actual class	Defaulter	[redacted]	[redacted]
	Delayer	[redacted]	[redacted]

(b) Confusion matrix with minimal cost. Threshold is 0.178. Total cost is [redacted].

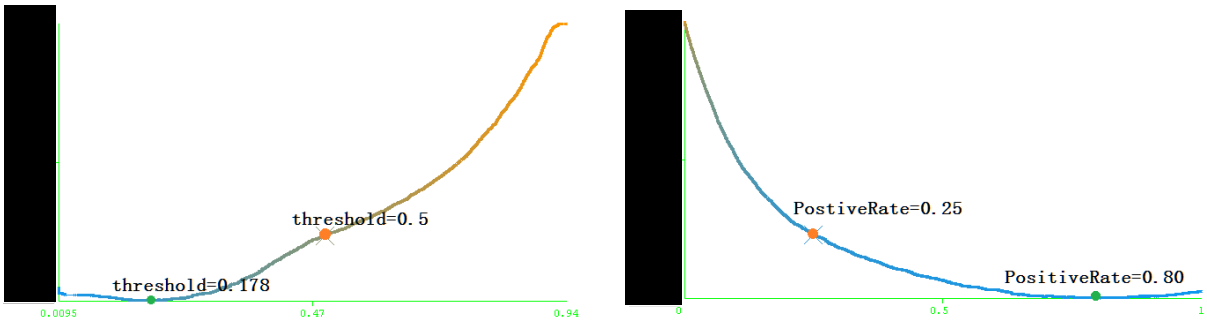
Table 6.7: Confusion matrices.

The cost curve plotted in figure 6.3 can inspire our deeper understanding. The threshold 0.178 gives us the minimal cost as the green point in the left figure, but the corresponding positive rate is around 0.80 as the green point in the right figure. In other word, 80% of the new arrear customers are classified as defaulters. Although the total cost is smallest, is it a smart decision at the moment? Obviously not, because the Arrears department does not have enough capacity of case managers to handle (call) 80% of the new arrear customers[1]. It is noted that the cost curve in figure 6.3b is monotonically decreasing before the lowest cost point (the green point), so in the real deployment of the model the best threshold corresponds to the maximum capacity of case managers. The positive rate of our balanced random forests model is around 25% to 30%, which is fitting the current contacting ability of the Arrears department.

6.3 Random forests feature importance analysis

As mentioned in subsection 2.3, an embedded method is the third way to select features apart from the filter and wrapper approaches. Some popular embedded methods are random forests/decision trees feature importance, weight vector of SVM, neural networks coefficients [83]. Since we have already

[1] However, it must be noted here that the capacity constraint is just at the moment. This does provide valuable insight for the future when decisions need to be made about capacity.



(a) (b)

Figure 6.3: Cost curve plotting. (a) X-axis is the cut-off threshold. Y-axis is the total cost. The green point on which the cut-off threshold is 0.178 corresponds the minimal cost; the red point represents the default cut-off threshold 0.5 of balanced random forests, which gives more cost. (b) X-axis is the percentage of predicted defaulters of all new arrear customers. Y-axis is the total cost. The green point gives the minimal cost (corresponds to the green point in the left figure), where our model classifies 80% customers as defaulters. Although the default cut-off threshold (the red point) gives more cost, the percentage of predicted defaulters (25%) matches the current capacity of the case managers of the Arrears department.

determined balanced random forests as our model, it is natural to employ built-in functionalities of random forests to analyse the data further.

The calculation of the feature importance accompanies with the procedure of building the forests as shown in algorithm 8. We run balanced random forests again with $N_{trees} = 2000$ and $m_{try} = 70$, and plot the top 30 features in figure 6.4. Some interesting points can be summarized:

- [redacted]
- [redacted]
- [redacted]

[redacted] terms are [redacted]

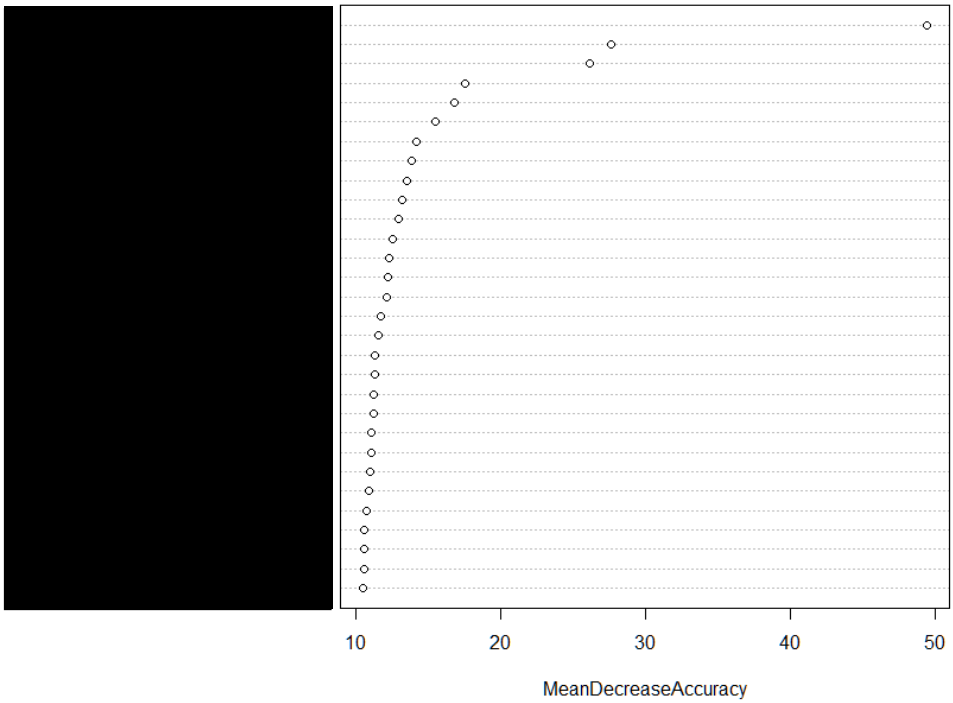


Figure 6.4: Dot chart of feature importance. The Y-axis shows the name of features. They are ordered top-to-bottom as most-to-least important. The X-axis is the mean decrease in accuracy as determined during the out of bag error calculation phase. The more the accuracy of the random forests decreases due to the addition of a single variable, the more important the variable is deemed, and therefore variables with a large mean decrease in accuracy are more important for classification of the data.

Algorithm 8 Feature importance of random forests

Input: training set S with P features, integer M (number of trees)
for $i = 1$ to M **do**
    Construct the ith tree
    The OOB (out-of-bag) samples are put down into the tree, and calculate the accuracy
    for $j = 1$ to P **do**
        Randomly permuted jth feature in OOB samples
        Put down the "new" OOB samples again and calculate the accuracy
        Record the reduction of accuracy
    end for
end for
Output: Average of the reduction of accuracy over all M trees, used as a measure of importance of features.

Test with top 30 features Another fact from figure 6.4 is that the feature importance decreases dramatically from DLT, the third most important feature. Then, the importance keeps more and more stable. We make a new experiment with just running a 10-fold cross validation with the top 30 features and with $N_{tree} = 1000$, $m_{try} = \sqrt{30} = 5$(default value). The AUC value is 0.79681 ± 0.0110. Not surprisingly, the result is better than when using $N_{tree} = 1000$ and $m_{try} = 5$ on all features (0.78950 ± 0.0115, if we look up table 6.9). If we test with $N_{tree} = 1000$, $m_{try} = 20$, the result is 0.80106 ± 0.0116, which is even better than the best result 0.80016 ± 0.0130 but not significantly (p-value is 0.56416). Some random forests packages such as Fortran code implemented by Breiman and Cutler[3] utilize the result of feature importance when building the forest: "if the number of variables is very large, forests can be run once with all the variables, then run again using only the most important features from the first run".

6.4 Individual variables analysis by logistic regression coefficients

We can also interpret logistic regression coefficients. Let us paste the regression equation here.

$$\ln(\frac{p_i}{1-p_i}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n \tag{6.4}$$

Dummy variables have already been explained in section 3.3. First, all numeric features are discretized to nominal features by the Entropy Minimum Description Length principle. Then, nominal features are binarized into dummy variables, e.g. x_1 in equation 6.4 is converted to $x_{11}, x_{12}, \cdots, x_{1C}$ in equation 6.5, which are either 0 or 1.

$$\ln(\frac{p_i}{1-p_i}) = \beta_0 + \underbrace{\beta_{11}x_{11} + \beta_{12}x_{12} + \cdots + \beta_{1C_1}x_{1C_1}}_{\beta_1 x_1} + \cdots + \underbrace{\beta_{n1}x_{n1} + \beta_{n2}x_{n2} + \cdots + \beta_{nC_n}x_{nC_n}}_{\beta_n x_n} \tag{6.5}$$

$\ln(\frac{p_i}{1-p_i})$ is a monotonic increasing function, hence the sign and absolute value of coefficient β of dummy variables decide how the dummy variable impacts on classification: if the coefficient is a positive number, the corresponding dummy variable is prone to defaulters, and a negative coefficient to delayers; the magnitude indicates the extent of tendentiousness.

Table 6.8 lists dummy variable coefficients of some features. [redacted].

[3] http://www.stat.berkeley.edu/~breiman/RandomForests/cc_software.htm

Features	Cutoff region of dummy variable	Coefficient
[redacted]	$(-\infty, 0.15]$	-0.6533
	$(0.15, 2.85]$	0.3617
	$(2.85, \infty)$	0.6634
[redacted]	$(-\infty, 0.000409]$	-0.315
	$(0.000409, 0.000509]$	-0.2293
	$(0.000509, 0.002663]$	0.0201
	$(0.002663, 0.023208]$	0.2426
	$(0.023208, \infty)$	0.1543
[redacted]	$(-\infty, 61]$	0.1069
	$(61, 3001]$	-0.0176
	$(3001, \infty]$	-0.0509
[redacted]	$(-\infty, 16.8]$	0.1586
	$(16.8, 1045]$	-0.0038
	$(1045, 3139]$	-0.1287
	$(3139, \infty]$	-0.3313
[redacted]	$(-\infty, -825]$	0.0087
	$(-825, 970]$	-0.0057
	$(970, \infty]$	-0.1606
[redacted]	$(-\infty, 438]$	0.1887
	$(438, 11538]$	-0.0989
	$(11538, \infty]$	-0.5049

Table 6.8: Dummy variables and coefficient. The second column lists some numeric features; the third column is the discrete region divided by cut-off points; the fourth column is the corresponding coefficient.

m_{try} \ N_{tree}	**50**	**100**	**200**	**300**	**400**
1	0.75168 ± 0.0095	0.75877 ± 0.0132	0.75873 ± 0.0105	0.76222 ± 0.0108	0.76184 ± 0.0109
5	0.77410 ± 0.0098	0.78322 ± 0.0085	0.78542 ± 0.0097	0.78763 ± 0.0108	0.78861 ± 0.0102
10 (default)	0.78431 ± 0.0088	0.78557 ± 0.0078	0.79150 ± 0.0080	0.79212 ± 0.0077	0.79218 ± 0.0079
20	0.78626 ± 0.0132	0.79215 ± 0.0111	0.79635 ± 0.0112	0.79602 ± 0.0112	0.79700 ± 0.0108
30	0.78559 ± 0.0132	0.79329 ± 0.0098	0.79606 ± 0.0107	0.79714 ± 0.0133	0.79768 ± 0.0124
40	0.78946 ± 0.0108	0.79505 ± 0.0131	0.79717 ± 0.0118	0.79868 ± 0.0126	0.79828 ± 0.0130
50	0.79050 ± 0.0133	0.79536 ± 0.0145	0.79720 ± 0.0124	0.79766 ± 0.0133	0.79735 ± 0.0125
60	0.79042 ± 0.0119	0.79504 ± 0.0113	0.79778 ± 0.0123	0.79774 ± 0.0137	0.79844 ± 0.0120
70	0.78979 ± 0.0130	0.79587 ± 0.0145	0.79684 ± 0.0130	0.79790 ± 0.0129	0.79819 ± 0.0127
80	0.79028 ± 0.0144	0.79540 ± 0.0127	0.79702 ± 0.0124	0.798803 ± 0.0131	0.79892 ± 0.0127

m_{try} \ N_{tree}	**500**	**750**	**1000**	**1500**	**2000**
1	0.76183 ± 0.0102	0.76225 ± 0.0114	0.76333 ± 0.0117	0.76340 ± 0.0110	0.76348 ± 0.0111
5	0.78966 ± 0.0114	0.79006 ± 0.0109	0.78950 ± 0.0115	0.78934 ± 0.0105	0.79058 ± 0.0101
10 (default)	0.79342 ± 0.0086	0.79339 ± 0.0114	0.79375 ± 0.0086	0.79435 ± 0.0114	0.79397 ± 0.0085
20	0.79584 ± 0.0127	0.79745 ± 0.0116	0.79822 ± 0.0129	0.79862 ± 0.0121	0.79773 ± 0.0122
30	0.79783 ± 0.0125	0.79881 ± 0.0124	0.79891 ± 0.0128	0.79866 ± 0.0127	0.79864 ± 0.0123
40	0.79892 ± 0.0125	0.79931 ± 0.0125	0.79967 ± 0.0126	0.79937 ± 0.0129	0.79942 ± 0.0126
50	0.79817 ± 0.0122	0.79921 ± 0.0130	0.79922 ± 0.0126	0.79929 ± 0.0128	0.79966 ± 0.0128
60	0.79970 ± 0.0131	0.79920 ± 0.0127	0.79960 ± 0.0130	0.79955 ± 0.0130	0.79990 ± 0.0131
70	0.79962 ± 0.0127	0.79924 ± 0.0136	0.79973 ± 0.0128	0.79977 ± 0.0128	0.80016 ± 0.0130
80	0.79875 ± 0.0134	0.79992 ± 0.0132	0.79959 ± 0.0127	0.79997 ± 0.0129	0.79980 ± 0.0125

Table 6.9: The AUC values of different numbers of trees (N_{trees}) and numbers of variables randomly sampled as candidates at each split (m_{try}).

Chapter 7

Conclusion and future work

7.1 Conclusion

Now, we can start answering the research questions in the first chapter:

1. Can we build a binary classification model to distinguish defaulters and delayers from new arrear customers at the beginning of each month?
2. If the answer to question 1 is positive, can the model also have good interpretability so that some rules, patterns or useful discovery can be used by the Arrears Department?

We chose balanced random forests as the classification model. The model gives an AUC value of around 0.7856 during testing. When the model was deployed in the daily work of the Arrears department of ING, we know the real labels of predicted new arrear customers at the end of the month. The real labels can be used as validation. The AUC value of May 2013 is 0.7714, which is promising and consistent with the test result. Useful knowledge is also discovered, such as feature importance, cost matrix analysis and logistic regression coefficient analysis. They can guide the daily work of the Arrears department and give a deep insight to this problem.

The classification model just assigns the labels of "defaulters" or "delayers" to the new arrear customers. In order to apply the power of classification, a new process must be formulated to treat new arrear customers differentially. Figure 7.2 demonstrates it roughly. Comparing with the old process in figure 7.1, we can see that the case managers get involved at the very beginning for the defaulters (red customers), meanwhile the delayers (green customers) are treated automatically by Emails, letters and SMS.

We started to test the model and new process in the Arrears department from [redacted] Since [redacted], the model and new process were deployed officially.

Compared to the old process, the new process can push 19% $\sim$ 30% more defaulters out of arrears[1]. Then, we can estimate the cost saving by deploying the model and new process: In 2013, the Arrears department has around [redacted] defaulters per month and our model can target more than 60% of the defaulters, which means [redacted] defaulters will be treated by case managers intensively; then [redacted] is multiplied by the recovery rate mentioned above, let us use 20% which is the lower bound in the range, we can derive that [redacted] defaulters go back healthy approximately. Appendix A shows that one defaulter will bring interest loss, provision loss and potential loss. If we only account the one-term provision loss

[1] The way to compare the old and new process is like this: select two groups of defaulters with the same size. One group of defaulters is treated by the old process, i.e., only letter and SMS, while another group is treated with a new treatment, i.e., intensive phone call contact. At the end of the month, count and compare how many defaulters are out of arrears in the two groups.

[redacted] conservatively, **our model and new process will save ING [redacted] per month!**

This project is one of the pioneer projects which import the power of classification into daily banking operation. Inspired by the success of this project, ING firms the strategy to employ machine learning and big data.

7.2 Future work

We outline future work for improving the proposed model:

Gradient Boosting Proposed by Friedman in [86], gradient boosting might be an interesting classifier to our problem, or at least a good basic classifier in the ensemble by voting or stacking, because it usually gets similar or even better results than random forests [87]. Gradient boosting builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function. The same as random forests, gradient boosting can be used to carry out variable selection during the training process. In addition, partial dependence can be generated as well to show the dependence between the target response and a set of "target" features, marginalizing over the values of all other features (the "complement" features).

Ensemble Selection In section 5.3, we simply combined the best four learners by voting and stacking and the result did not outperform basic classifiers significantly. It is a kind of trying our luck because just picking up four best standalone classifiers cannot ensure that the ensemble of them improves remarkably. Caruana proposed "ensemble selection from libraries of models" in [88], which should obtain a better model. The basic idea is building a library of models with approximately hundreds or thousands of models. All classifiers with all different parameters will be included in the library. Then a hillclimbing searching strategy is employed to select and ensemble classifiers. Algorithm 9 shows the detail. Ensemble selection has been highlighted in winning solutions of the Netflix prize [89] and the KDD Cup 2009 [90].

Algorithm 9 Ensemble Selection.

Input: Start with the empty ensemble and a library of models trained before.

1. Add to the ensemble the model in the library that maximizes the ensemble's performance to the error metric on a hillclimb (validation) set.
2. Repeat Step 2 for a fixed number of iterations or until all the models have been used.

Output: the ensemble from the nested set of ensembles that has maximum performance on the hillclimb (validation) set.

Figure 7.1: Old Process. All customers are treated indistinctly.

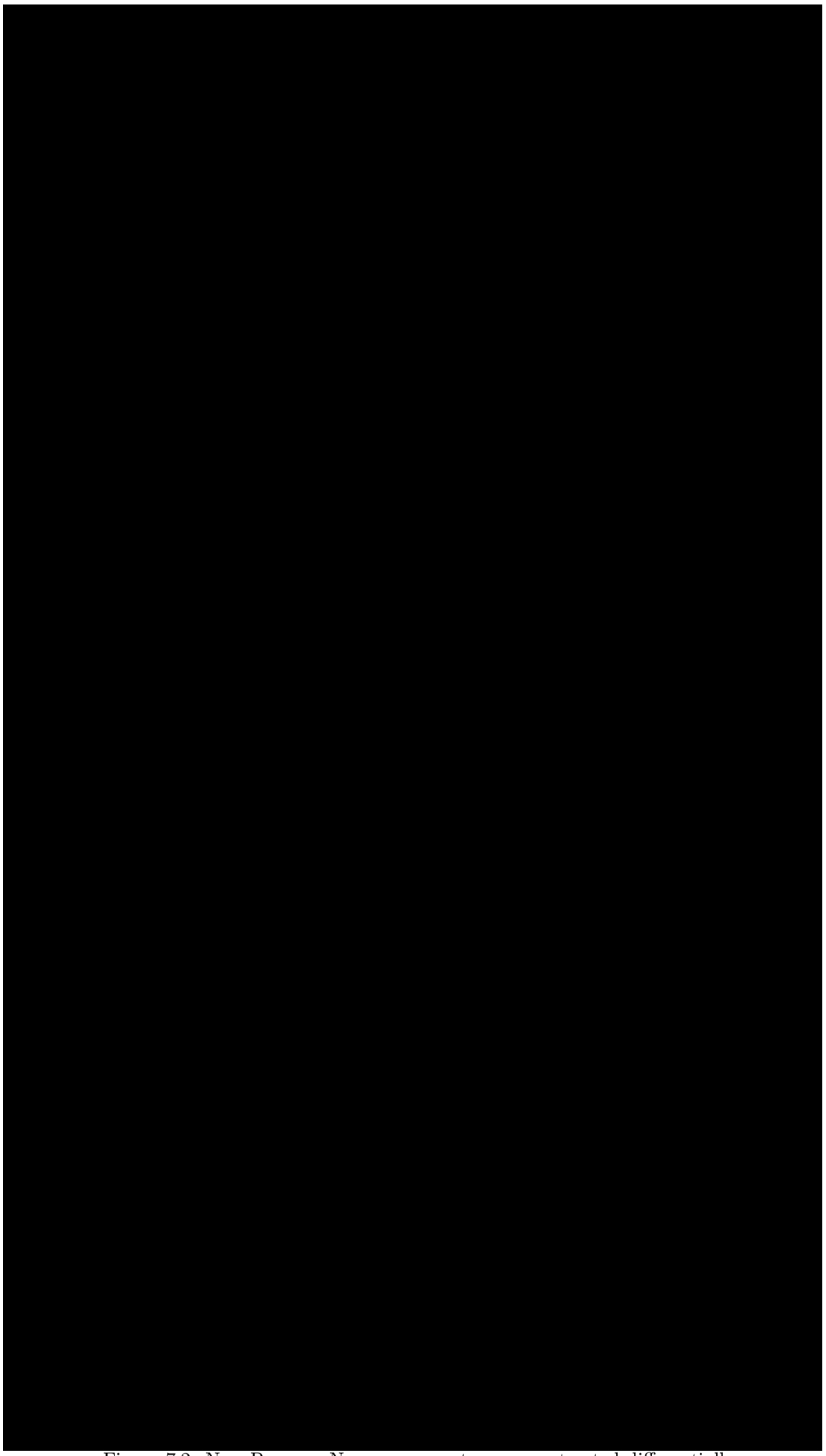


Figure 7.2: New Process. New arrear customers are treated differentially.

Appendices

Appendix A

Cost matrix

We will revisit the cost matrices and explain them in this appendix. Let us have a look at the operational cost matrix in table A.2a first.

Operational cost matrix Automatic treatments like Email/SMS/letter will be sent to all new customers no matter defaulters or delayers. By merely calculating the cost per Email, SMS and letter, we get [redacted] per new arrear customer. Defaulters have extra treatments by phone calls. We calculate the cost of phone calls as below:

1. Count the yearly salary costs of case managers and all overhead;
2. Divided by number of effective hours per year (total effective hours per day $\times$ working days per month $\times$ number of months), we get the cost per effective hour.
3. Divided by the average number of phone calls per effective hour, we get the cost of a phone call treatment per defaulters.

The calculation gives around [redacted] per defaulter. In this way, we get the numbers in table A.2a.

Processing benefit matrix The risk cost mainly contains three parts:

- Interest loss: because customers do not pay monthly debt on time, ING will lose interest of the debt. Suppose average monthly debt per new arrear customer is [redacted] and interest rate is [redacted] per month [91], the interest loss is [redacted]
- Potential collection loss: if a customer misses the payment longer than 6 months, his/her property will be sold or auctioned by force. The price is lower than the normal real estate market, so it brings ING the loss. The average loss by default is [redacted]. Of course, only a small fraction of new arrear customers will be in this phase.
- Provision: as in figure 1.2a, if a mortgage customer has arrears, a certain amount of potential loss is frozen as provision. The potential loss is around [redacted] per year. If the customer is found one term debt delay, ING needs to freeze [redacted] in the next month. Similarly, two terms debt delay leads to [redacted] and missing more than three terms leads to [redacted].

Here, we consider processing benefit as the risk cost saving after employing the new process, rather than the absolute value of risk cost. Hence, table A.1 lists the decreasing risk cost. Some explanations are following:

- Predicted delayer (first row): no matter who the new arrear customer is, he/she will get the loose treatment (Email/SMS/letter), which is just the same as in the old process. So, no risk cost changes.
- Predicted defaulter who is an actual delayer (second row): although the new arrear customer receives intensive treatment, he/she is an actual delayer and will repay the debt before the end of the month anyway. So, the risk cost does not change either.
- Predicted defaulter who is an actual defaulter (third row): this defaulter is correctly spotted and treated. But we should notice that even a proper treatment cannot help defaulters sometimes. So,

we import four coefficients α, β_1, β_2 and θ. By analysing the experimental deployment result of March, April and May 2013, α is around [redacted] For example, if there are [redacted] defaulters who are treated by the old process and go back healthy, then [redacted] defaulters can go back healthy with the new process. β_1 and β_2 are still unknown. θ is around [redacted]. Suppose β_1 and β_2 are both [redacted] is a conservative estimation.

Type of new arrear customers	Interest loss	Provision	Potential collection loss	In total
Predicted delayer	[redacted]	[redacted]	[redacted]	[redacted]
Predicted defaulter, actual delayer	[redacted]	[redacted]	[redacted]	[redacted]
Predicted defaulter, actual defaulter	[redacted]	[redacted]	[redacted]	[redacted]

Table A.1: Decreasing risk cost by employing new process. α is percentage extra customers to get healthy with the new treatment; β_1 and β_2 are how many less percentages of defaulters miss two and three or longer than three terms after the treatment; θ is how many less percentages of defaulters miss longer than six terms after the treatment.

		Predict class	
		Defaulter	Delayer
Actual class	Defaulter	[redacted]	[redacted]
	Delayer	[redacted]	[redacted]

(a) Operational cost matrix.

		Predict class	
		Defaulter	Delayer
Actual class	Defaulter	[redacted]	[redacted]
	Delayer	[redacted]	[redacted]

(b) Processing benefit matrix.

Table A.2: Operational cost matrix and processing benefit matrix. Units of the numbers in the matrices are Euro.

Bibliography

[1] ING Group, "Ing group fact sheet," 2012.

[2] H. van der Noordaa, "Retail banking benelux."

[3] W. Vandevyvere and A. Zenthöfer, "The housing market in the Netherlands," tech. rep., Directorate General Economic and Monetary Affairs, European Commission, 2012.

[4] J. A. Chopoorian, R. Witherell, O. E. Khalil, and M. Ahmed, "Mind your business by mining your data," *SAM Advanced Management Journal*, vol. 66, no. 2, pp. 45–51, 2001.

[5] S.-H. Liao, P.-H. Chu, and P.-Y. Hsiao, "Data mining techniques and applications–a decade review from 2000 to 2011," *Expert Systems with Applications*, vol. 39, no. 12, pp. 11303–11311, 2012.

[6] A. M. Hormozi and S. Giles, "Data mining: a competitive weapon for banking and retail industries," *Information systems management*, vol. 21, no. 2, pp. 62–71, 2004.

[7] P. Domingos, "A few useful things to know about machine learning," *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, 2012.

[8] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *The Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[9] S. Cateni, M. Vannucci, and V. Colla, "Variable selection and feature extraction through artificial intelligence techniques," 2012.

[10] R. N. Khushaba, A. Al-Ani, and A. Al-Jumaily, "Differential evolution based feature subset selection," in *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pp. 1–4, IEEE, 2008.

[11] Z. Xiao, E. Dellandrea, W. Dou, and L. Chen, "ESFS: A new embedded feature selection method based on SFS," *Rapports de recherché*, 2008.

[12] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial intelligence*, vol. 97, no. 1, pp. 273–324, 1997.

[13] Y. Saeys, T. Abeel, and Y. Van de Peer, "Robust feature selection using ensemble feature selection techniques," in *Machine Learning and Knowledge Discovery in Databases*, pp. 313–325, Springer, 2008.

[14] Q. Shen, R. Diao, and P. Su, "Feature selection ensemble," in *Proceedings of the Alan Turing centenary conference*, 2012.

[15] B. Waad, B. M. Ghazi, and L. Mohamed, "Rank aggregation for filter feature selection in credit scoring," *International Conference on Control, Engineering and Information Technology (CEIT 13)*, vol. 1, pp. 64–68, 2013.

[16] F. Mörchen, "Time series feature extraction for data mining using DWT and DFT," *Citeseer*, 2003.

[17] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," in *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pp. 2–11, ACM, 2003.

[18] E. J. Keogh and M. J. Pazzani, "Scaling up dynamic time warping for datamining applications," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 285–289, ACM, 2000.

[19] V. Chandola, A. Banerjee, and V. Kumar, "Outlier detection: A survey," 2007.

[20] H. Liu, F. Hussain, C. L. Tan, and M. Dash, "Discretization: An enabling technique," *Data mining and knowledge discovery*, vol. 6, no. 4, pp. 393–423, 2002.

[21] E. I. Altman, "Financial ratios, discriminant analysis and the prediction of corporate bankruptcy," *The journal of finance*, vol. 23, no. 4, pp. 589–609, 1968.

[22] B. Baesens, T. Van Gestel, S. Viaene, M. Stepanova, J. Suykens, and J. Vanthienen, "Benchmarking state-of-the-art classification algorithms for credit scoring," *Journal of the Operational Research Society*, vol. 54, no. 6, pp. 627–635, 2003.

[23] L. C. Thomas, "A survey of credit and behavioural scoring: forecasting financial risk of lending to consumers," *International Journal of Forecasting*, vol. 16, no. 2, pp. 149–172, 2000.

[24] D. Feldman and S. Gross, "Mortgage default: classification trees analysis," *The Journal of Real Estate Finance and Economics*, vol. 30, no. 4, pp. 369–396, 2005.

[25] Q. Gan, B. Luo, and Z. Lin, "Risk management of residential mortgage in China using date mining a case study," in *New Trends in Information and Service Science, 2009. NISS'09. International Conference on*, pp. 1378–1383, IEEE, 2009.

[26] M. Malik and L. C. Thomas, "Transition matrix models of consumer credit ratings," *International Journal of Forecasting*, vol. 28, no. 1, pp. 261–272, 2012.

[27] N.-C. Hsieh, "An integrated data mining and behavioral scoring model for analyzing bank customers," *Expert systems with applications*, vol. 27, no. 4, pp. 623–633, 2004.

[28] P. P. Bonissone and W. Cheetham, "Financial applications of fuzzy case-based reasoning to residential property valuation," in *Fuzzy Systems, 1997, Proceedings of the Sixth IEEE International Conference on*, vol. 1, pp. 37–44, IEEE, 1997.

[29] G. H. Lee, "Rule-based and case-based reasoning approach for internal audit of bank," *Knowledge-Based Systems*, vol. 21, no. 2, pp. 140–147, 2008.

[30] S. L. Mansar, F. Marir, and H. A. Reijers, "Case-based reasoning as a technique for knowledge management in business process redesign," *Electronic Journal on Knowledge Management*, vol. 1, no. 2, pp. 113–124, 2003.

[31] C.-S. Park and I. Han, "A case-based reasoning with the feature weights derived by analytic hierarchy process for bankruptcy prediction," *Expert Systems with Applications*, vol. 23, no. 3, pp. 255–264, 2002.

[32] S. Ho Ha and R. Krishnan, "Predicting repayment of the credit card debt," *Computers & Operations Research*, vol. 39, no. 4, pp. 765–773, 2012.

[33] S. H. Ha, "Behavioral assessment of recoverable credit of retailers customers," *Information Sciences*, vol. 180, no. 19, pp. 3703–3717, 2010.

[34] N. Capon, "Credit scoring systems: a critical analysis," *The Journal of Marketing*, pp. 82–91, 1982.

[35] A. Aamodt and E. Plaza, "Case-based reasoning: Foundational issues, methodological variations, and system approaches," *AI communications*, vol. 7, no. 1, pp. 39–59, 1994.

[36] J. C. Wiginton, "A note on the comparison of logit and discriminant models of consumer credit behavior," *Journal of Financial and Quantitative Analysis*, vol. 15, no. 03, pp. 757–770, 1980.

[37] X.-L. Li and Y. Zhong, "An overview of personal credit scoring: Techniques and future work," *International Journal*, vol. 2, 2012.

[38] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, S. Y. Philip, *et al.*, "Top 10 algorithms in data mining," *Knowledge and Information Systems*, vol. 14, no. 1, pp. 1–37, 2008.

[39] T. M. Mitchell, "Machine learning. 1997," *Burr Ridge, IL: McGraw Hill*, vol. 45, 1997.

[40] M. A. Hall, *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato, 1999.

[41] W.-Y. Loh, "Classification and regression trees," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 14–23, 2011.

[42] H. He and E. A. Garcia, "Learning from imbalanced data," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 21, no. 9, pp. 1263–1284, 2009.

[43] M. P. Brown, W. N. Grundy, D. Lin, N. Cristianini, C. Sugnet, M. Ares, and D. Haussler, "Support vector machine classification of microarray gene expression data," *University of California, Santa Cruz, Technical Report UCSC-CRL-99-09*, 1999.

[44] C.-T. Su and Y.-H. Hsiao, "An evaluation of the robustness of MTS for imbalanced data," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 19, no. 10, pp. 1321–1332, 2007.

[45] Y. Sun, M. S. Kamel, and Y. Wang, "Boosting for learning multiple classes with imbalanced class distribution," in *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pp. 592–602, IEEE, 2006.

[46] Z.-H. Zhou and X.-Y. Liu, "On multi-class cost-sensitive learning," *Computational Intelligence*, vol. 26, no. 3, pp. 232–257, 2010.

[47] N. Abe, B. Zadrozny, and J. Langford, "An iterative method for multi-class cost-sensitive learning," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 3–11, ACM, 2004.

[48] G. E. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 20–29, 2004.

[49] T. Fawcett, "ROC graphs: Notes and practical considerations for researchers," *Machine Learning*, vol. 31, pp. 1–38, 2004.

[50] F. Provost and P. Domingos, "Well-trained pets: Improving probability estimation trees," 2000.

[51] P. Domingos, "Metacost: a general method for making classifiers cost-sensitive," in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 155–164, ACM, 1999.

[52] G. M. Weiss, "Mining with rarity: a unifying framework," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 7–19, 2004.

[53] S. Kotsiantis, D. Kanellopoulos, P. Pintelas, *et al.*, "Handling imbalanced datasets: A review," *GESTS International Transactions on Computer Science and Engineering*, vol. 30, no. 1, pp. 25–36, 2006.

[54] Y. Sun, A. K. Wong, and M. S. Kamel, "Classification of imbalanced data: A review," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 23, no. 04, pp. 687–719, 2009.

[55] C. Lemnaru and R. Potolea, "Imbalanced classification problems: systematic study, issues and best practices," in *Enterprise Information Systems*, pp. 35–50, Springer, 2012.

[56] X. Guo, Y. Yin, C. Dong, G. Yang, and G. Zhou, "On the class imbalance problem," in *Natural Computation, 2008. ICNC'08. Fourth International Conference on*, vol. 4, pp. 192–201, IEEE, 2008.

[57] H. He and Y. Ma, *Imbalanced Learning: Foundations, Algorithms, and Applications*. Wiley. com, 2013.

[58] R. C. Holte, L. Acker, and B. W. Porter, "Concept learning and the problem of small disjuncts.," in *IJCAI*, vol. 89, pp. 813–818, 1989.

[59] C. Drummond, R. C. Holte, *et al.*, "C4. 5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling," in *Workshop on Learning from Imbalanced Datasets II*, vol. 11, 2003.

[60] I. Mani and I. Zhang, "KNN approach to unbalanced data distributions: a case study involving information extraction," in *Proceedings of Workshop on Learning from Imbalanced Datasets*, 2003.

[61] X.-Y. Liu, J. Wu, and Z.-H. Zhou, "Exploratory undersampling for class-imbalance learning," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 39, no. 2, pp. 539–550, 2009.

[62] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, no. 1, pp. 321–357, 2002.

[63] H. Han, W.-Y. Wang, and B.-H. Mao, "Borderline-smote: A new over-sampling method in imbalanced data sets learning," in *Advances in intelligent computing*, pp. 878–887, Springer, 2005.

[64] E. Ramentol, Y. Caballero, R. Bello, and F. Herrera, "Smote-rsb*: a hybrid preprocessing approach based on oversampling and undersampling for high imbalanced data-sets using SMOTE and rough sets theory," *Knowledge and Information Systems*, vol. 33, no. 2, pp. 245–265, 2012.

[65] N. V. Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer, "Smoteboost: Improving prediction of the minority class in boosting," in *Knowledge Discovery in Databases: PKDD 2003*, pp. 107–119, Springer, 2003.

[66] L. Lusa *et al.*, "Smote for high-dimensional class-imbalanced data," *BMC bioinformatics*, vol. 14, no. 1, pp. 1–16, 2013.

[67] C. Bolton, *Logistic regression and its application in credit scoring*. PhD thesis, University of Pretoria, 2009.

[68] B. V. Dasarathy and B. V. Sheela, "A composite classifier system design: concepts and methodology," *Proceedings of the IEEE*, vol. 67, no. 5, pp. 708–713, 1979.

[69] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[70] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[71] T. K. Ho, "The random subspace method for constructing decision forests," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 20, no. 8, pp. 832–844, 1998.

[72] R. E. Schapire, "The strength of weak learnability," *Machine learning*, vol. 5, no. 2, pp. 197–227, 1990.

[73] Y. Freund, R. E. Schapire, *et al.*, "Experiments with a new boosting algorithm," in *ICML*, vol. 96, pp. 148–156, 1996.

[74] J. H. Friedman, "Stochastic gradient boosting," *Computational Statistics & Data Analysis*, vol. 38, no. 4, pp. 367–378, 2002.

[75] A. Liaw and M. Wiener, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[76] M. Kearns and L. Valiant, "Cryptographic limitations on learning boolean formulae and finite automata," *Journal of the ACM (JACM)*, vol. 41, no. 1, pp. 67–95, 1994.

[77] Y. Freund and R. E. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," in *Computational learning theory*, pp. 23–37, Springer, 1995.

[78] R. C. Holte, "Very simple classification rules perform well on most commonly used datasets," *Machine learning*, vol. 11, no. 1, pp. 63–90, 1993.

[79] M. Galar, A. Fernández, E. Barrenechea, H. Bustince, and F. Herrera, "A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 42, no. 4, pp. 463–484, 2012.

[80] S. Hido, H. Kashima, and Y. Takahashi, "Roughly balanced bagging for imbalanced data," *Statistical Analysis and Data Mining*, vol. 2, no. 5-6, pp. 412–426, 2009.

[81] C. Chen, A. Liaw, and L. Breiman, "Using random forest to learn imbalanced data," *University of California, Berkeley*, 2004.

[82] E. Alpaydin, *Introduction to machine learning*. MIT press, 2004.

[83] Y. Saeys, I. Inza, and P. Larrañaga, "A review of feature selection techniques in bioinformatics," *bioinformatics*, vol. 23, no. 19, pp. 2507–2517, 2007.

[84] R. B. Avery, R. W. Bostic, P. S. Calem, and G. B. Canner, "Credit risk, credit scoring, and the performance of home mortgages," *Fed. Res. Bull.*, vol. 82, p. 621, 1996.

[85] J. W. Straka, "A shift in the mortgage landscape: The 1990s move to automated credit evaluations," *Journal of Housing Research*, vol. 11, no. 2, pp. 207–232, 2000.

[86] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of Statistics*, pp. 1189–1232, 2001.

[87] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer Series in Statistics, New York, NY, USA: Springer New York Inc., 2001.

[88] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes, "Ensemble selection from libraries of models," in *Proceedings of the twenty-first international conference on Machine learning*, p. 18, ACM, 2004.

[89] J. Bennett and S. Lanning, "The netflix prize," in *Proceedings of KDD cup and workshop*, vol. 2007, p. 35, 2007.

[90] M. Tavallaee, E. Bagheri, W. Lu, and A.-A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications 2009*, 2009.

[91] CPB Economic Policy Analysis, "The dutch housing market: mortgage interest rates, house prices and consumption," 2013.