CHAPTER 2
BACKGROUND KNOWLEDGE

This chapter covers the background knowledge of this dissertation which contains classifiers used in this dissertation, class imbalance problem and groups of strategies for dealing with the problem and performance measures used in this dissertation. In this chapter, one strategy which is to use oversampling techniques on imbalanced datasets during the data preprocessing stage is highlighted and several existing oversampling techniques are described.

2.1 Classification models

Normally, classifiers can be categorized as the supervised learning. All classifiers require a dataset which contains one attribute (mostly nominal) as a target or a class with several distinct values. Then, classifiers learn patterns, rules or characteristics in a given dataset that can categorize instances into different classes. Methods to learn these patterns, rules or characteristics are different depending on each algorithm. Users can use these patterns or rules given by algorithms to classify instances with the unknown class in the same problem. Data mining models which are widely used for classification problems are decision tree, naïve Bayes model, support vector machine, neural network and *k*-nearest neighbor.

2.1.1 Decision Tree

Decision tree [7] is a classification model which represents rules to recursively partition instances with hierarchical structures [35] [36]. The structure of decision tree contains leaves, each of which indicates a class and decision nodes which specify some evaluations to be carried out on a single attribute value with one branch and subtree for each possible outcome. A decision tree is used to classify an instance by starting at the root of the tree and moving through it until a leaf is encountered. At each non-leaf decision node, the outcome of each instance for each evaluation at the node is determined and it sends that instance to the root of the subtree corresponding to the outcome. When this process finally leads to a leaf, a class of the instance is predicted as the one at the leaf. The decision tree transforms a dataset into more concise form while the necessary characteristics of the dataset are still preserved. Moreover, it can disclose the relationship between independent attributes and given target classes which can be used to predict the target class for unlabeled instances in the future.

The task of constructing a tree from the training set has been called tree induction, tree building and tree growing. Most existing tree induction systems proceed in a greedy top-down fashion [37]. The tree construction starts with an empty tree and the entire training set at the root node. If all instances in a current node are in the same class, the leaf node of that class is created. Otherwise, a method to express attribute test conditions is chosen to find candidates of splits depending on the attribute types. Then, a goodness measure is used as a criterion to find the best split which separates instances into distinct multiple groups on each child node. The example of goodness measure is the entropy for ID3 [7] and C4.5 [32] and GINI index for CART [38]. The tree construction is performed recursively until every instance on each leaf node is from the same class. The resulting tree can be used for describing the dataset and predicting the target class of unknown instances. An example of a decision tree is shown in figure 2.

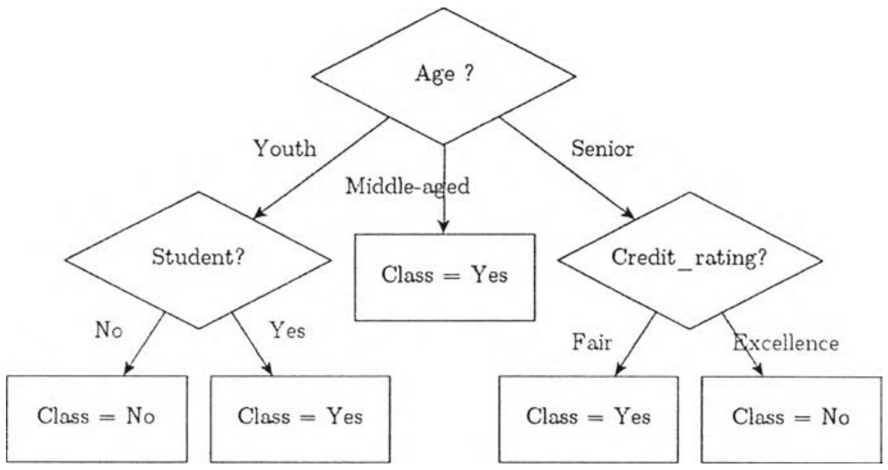


Figure 2: An example of decision tree

Decision tree algorithms are categorized based on the results of leaf nodes. The first one is a classification tree which provides the class as the discrete outcome to unknown instances if their attribute values are satisfied the conditions on each layer of the tree. The other one is a regression tree which provides the result as the real value number. Examples of decision tree algorithms are ID3 [7], C4.5 [32], C5.0 [39], CART [38], etc.

Using a decision tree as a classifier model has many advantages over other classification techniques [37]. The model is provided in the form of a tree which is easy to build, use and interpret. It does not require any complicated pre-processing procedures on the dataset to build a decision tree. Some decision tree algorithms can handle both numeric and categorical attributes. Finally, a decision tree induction algorithm is robust and able to handle large datasets. However, building an optimal decision tree is considered as an NP-Complete problem [40] [41]. Heuristics methods are required to find the optimal solution which is able to give only a local optimal solution not global. Some decision tree induction algorithms cause overfitting which require an additional process such as pruning [42] to avoid it.

2.1.2 Naïve Bayes Model for classification

A naïve Bayes model for classification [8] is a simple probabilistic model based on applying Bayes' theorem with the independence assumption. It assumes that the presence or absence of one attribute does not affect to the presence or absence of other attributes on the given target class. The naïve Bayes model considers each of these attributes to contribute independently to the probability, regardless of the presence or absence of other attributes.

Let $x = (x_1, x_2, \ldots, x_n)$ be an instance in a dataset where $x_1, x_2, \ldots, x_n$ are values of x in attributes $a_1, a_2, \ldots, a_n$, respectively. This dataset has k distinct class labels c_1, c_2, ..., c_k and the probability of each class label c_i in this dataset can be found. The posterior probability of class membership, i.e. the probability of x being in class c_i is written as $P(c_i \mid x_1, x_2, \ldots, x_n)$. Based on Bayes' theorem, this probability is:

$$P(c_i \mid x_1, x_2, \ldots, x_n) = \frac{P(c_i)P(x_1, x_2, \ldots, x_n \mid c_i)}{P(x_1, x_2, \ldots, x_n)}$$

Using the chain rule for repeated applications of the definition of conditional probability, $P(x_1, x_2, \ldots, x_n \mid c_i) = P(x_1 \mid c_i)\, P(x_2 \mid c_i, x_1) \ldots P(x_n \mid c_i, x_1, x_2, \ldots, x_{n-1})$. This term is equal to $P(x_1 \mid c_i) P(x_2 \mid c_i) \ldots P(x_n \mid c_i)$ or $\prod_{j=1}^{n} P(x_j \mid c_i)$ if the independence assumption is applied. For each $P(c_i \mid x_1, x_2, \ldots, x_n)$ where i = 1, 2, ..., k, $P(x_1, x_2, \ldots, x_n)$ is equal, so only $P(c_i) \cdot \prod_{j=1}^{n} P(x_j \mid c_i)$ is needed to be considered in place of the probability of x being in each class c_i as follows.

$$P(c_i \mid x_1, x_2, \ldots, x_n) = P(c_i) \cdot \prod_{j=1}^{n} P(x_j \mid c_i) \cdot \text{constant}$$

Then, the posterior probability for each class label c_i of x can be calculated. As the classification process, x is classified as the class label which provides the highest probability value. How to estimate each value of $P(x_j \mid c_i)$ depends on whether a_j is discrete or continuous. For a discrete attribute, the probability $P(x_j \mid c_i)$ is estimated from the ratio between the number of instances in class c_i which has the value of attribute a_j equal to x_j over the number of instances in class c_i. However, this estimation cannot be applied directly in the continuous attribute since there are too many distinct values in the continuous attribute. Several techniques suggested estimating the probability in continuous attributes such as discretizing the range into bins [43] and then representing each bin with one ordinal attribute or using a two-way split to bisect the value into two subsets, ones whose values are larger than the split point and ones whose values are smaller than the split point. However, these suggestions could violate the independence assumption. Another suggestion is to estimate the probability $P(x_j \mid c_i)$ by using probability density estimation. Under the assumption that the values in the attribute a_j form the normal distribution, parameters of distribution e.g., mean (μ_i) and standard deviation (σ_j) can be calculated. Once the probability distribution is known, it can be used to estimate each conditional probability $P(x_j \mid c_i)$ from the formula of normal (Gaussian) distribution as follows.

$$P(x_j \mid c_i) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(x_j - \mu_j / \sigma_j)^2}$$

After the conditional probability of each attribute is calculated, the posterior probability of $P(c_i \mid x)$ can be computed from the product of these conditional probabilities. However, if one of the probabilities is zero, the posterior probability becomes zero. In order to avoid this zero probability problem, the Laplace estimator or m-estimator is applied.

<u>Example</u>

Customer	Refund	Marital Status	Taxable Income	Evade
1.	Yes	Single	125k	No
2.	No	Married	100k	No
3.	No	Single	70k	No
4.	Yes	Married	120k	No
5.	No	Divorced	95k	Yes
6.	No	Married	60k	No
7.	Yes	Divorced	220k	No
8.	Yes	Single	85k	Yes
9.	No	Single	75k	No
10.	No	Married	90k	Yes

Given a unlabeled customer x = (Refund = No, Married, Income = 120k),

Then, P(x|No) = P(Refund = No|No) · P(Married|No) · P(Income = 120k|No).

Since P(Refund = No|No) = $\frac{4}{7}$, P(Married |No) = $\frac{3}{7}$ and for class = No, the sample mean of income is 110k and sample variance is 2,975,

$$\text{P(income = 120k | No)} = \frac{1}{\sqrt{2\pi(2975)}} e^{-\frac{1}{2}((120-110)/2975)^2} = 0.0072$$

Then P(x|No) = $\frac{4}{7} \times \frac{3}{7} \times (0.0072) = 0.0024$

On the other hand, P(x|Yes) = P(Refund = No|Yes) · P(Married|Yes) · P(Income = 120k|Yes).

Since P(Refund = No|Yes) = $\frac{2}{3}$, P(Married|No) = $\frac{1}{3}$ and for class = Yes, the sample mean of income is 90k and the sample variance is 25,

$$\text{P(income = 120k | Yes)} = \frac{1}{\sqrt{2\pi(25)}} e^{-\frac{1}{2}((120-90)/25)^2} = 1.2 \times 10^{-9}$$

Then P(x|Yes) = $\frac{2}{3} \times \frac{1}{3} \times (1.2 \times 10^{-9}) = 2.7 \times 10^{-10}$

Since P(x|No) · P(No) $= 0.0024 \times \frac{7}{10} > 2.7 \times 10^{-10} \times \frac{3}{10}$ = P(x|Yes) · P(Yes), P(No|x) > P(Yes|x) and x is classified as the class "No".

The naïve Bayes model has some advantages over other classifiers. Its algorithm is robust to outliers and it can handle missing values and irrelevant attributes. However, each attribute in the dataset has to hold the independence assumption and its continuous attributes are supposed to be the normal distribution in order to estimate conditional probabilities. Despite of these limitations, the naïve Bayes model for classification still is one of classification models widely applied on classification problems.

2.1.3 Support Vector Machine

Support Vector Machine (SVM) is a supervised learning model proposed by Boser, Guyon and Vapnik in 1992 [10] [44]. Its basic algorithm is designed to work on the dataset with two target classes by finding the linear hyperplane which is used as a decision boundary to linearly separate instances in the dataset into two sides; each represents one target class. In support vector machine, an instance is viewed as an n-dimensional vector. Then, a linear classifier is constructed if instances in the dataset can be separated into classes by a (n − 1)-dimensional hyperplane. There are many hyperplanes that could be used to classify the dataset, but this algorithm aims on finding the best hyperplane which represents the largest separation, or margin, between two classes. The chosen hyperplane is the one that maximizes the distance between it and the nearest data point on each side. If such a hyperplane exists, the hyperplane is known as the maximum-margin hyperplane and the linear classifier which defines this hyperplane is known as a maximum margin classifier.

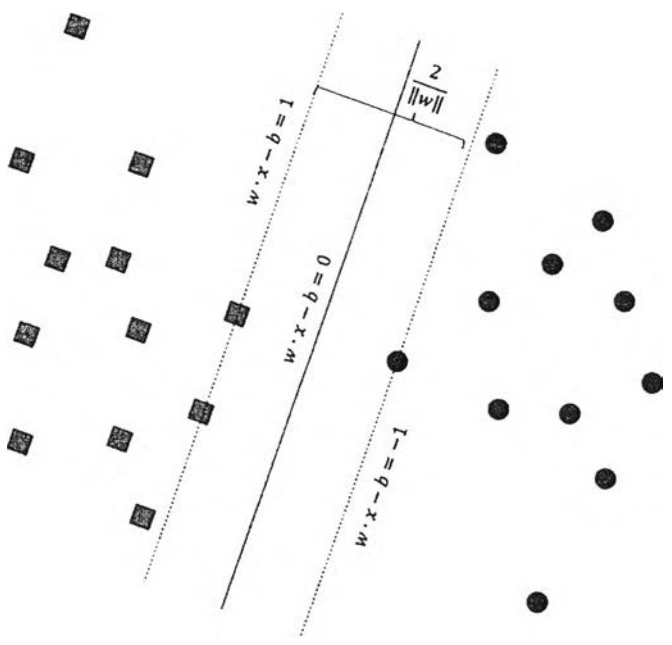


Figure 3: A visualization of simple support vector machine

Given an instance x_i in a dataset which contains m instances, n numerical attributes and the class label y_i which is either 1 or -1, indicating the class that x_i belongs to. Under the assumption that this dataset is linearly separable, the hyperplane which can separate all instances can be written as the set of instances satisfying $w{\cdot}x - b = 0$ where w is a normal vector of the hyperplane and $\frac{b}{\|w\|}$ determines the perpendicular distance to the hyperplane along the normal vector w. Then, if the dataset is linearly separable, instances can be separated by placing instances satisfying $w{\cdot}x - b \geq 1$ into one class and ones satisfying $w{\cdot}x - b \leq -1$ in another class. These conditions can be written as linear inequality constraints as $y_i(w{\cdot}x_i - b) \geq 1$ for $i = 1, 2, ..., m$. Given the region which separates instances is controlled by two hyperplanes $w{\cdot}x - b = 1$ and $w{\cdot}x - b = -1$, the margin between these two hyperplanes which equals to $\frac{2}{\|w\|}$ is needed to be maximized. Then this problem can be written as the optimization problem as follows.

$$\min_{w,b} \|w\|$$

$$\text{Subject to } \quad y_i(w{\cdot}x_i - b) \geq 1 \text{ for } i = 1, 2, ..., m \qquad (1)$$

This optimization problem is not easy to solve due to the square root under the norm $\|w\|$. However, it can be changed into quadratic programming optimization problem by transforming the objective function into $\frac{1}{2}\|w\|^2$.

Since the problem becomes the quadratic model, it can be changed into a Lagrangian formulation. The positive Lagrange multipliers are defined as α_i where i = 1, 2, ..., m for each inequality constraint. So (1) can be written as follows;

$$\arg \min_{w,b} \max_{\alpha \geqslant 0} \left\{\frac{1}{2}\|w\|^2 - \sum_{i=1}^{m} \alpha_i y_i (w \cdot x_i - b) + \sum_{i=1}^{m} \alpha_i\right\}$$

This problem can now be solved by standard quadratic programming techniques and programs. The "stationary" Karush–Kuhn–Tucker condition implies that the solution can be expressed as a linear combination of the training vectors/instances

$$w = \sum_{i=1}^{m} \alpha_i y_i x_i$$

where most Lagrange multipliers are set as zero, only a few α_i are greater than zero. The corresponding $x_i^{\bullet}$ with positive α_i are called support vectors, which lie on the margin and satisfy $y_i (w \cdot x_i^{\bullet} - b) = 1$. From this, it can be derived that support vectors satisfy $b = w \cdot x_i^{\bullet} - y_i$ which is used to define the offset b.

For datasets which cannot be separated completely with a single hyperplane, there is a method such as the soft margin [45] to select a hyperplane that can be classified instances with the least possible misclassified error while the margin between support vectors is still maximized. Slack variables ξ_i are added into constants to control the misclassified penalty of x_i. If the penalty function c is linear, then the previous optimization problem (1) becomes

$$\min_{w,\xi,b} \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{m} \xi_i$$

Subject to $\quad y_i (w \cdot x_i - b) \geq 1 - \xi_i \quad$ for i = 1, 2, ... , m (2)

Similarly with the previous problem, the Lagrangian formulation can be applied to (2) in order to solve for solution as follows;

$$\min_{w,\xi,b}\max_{\alpha,\beta}\left\{\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}\alpha_i\left[y_i(w\cdot x_i - b) - 1 + \xi_i\right] + \sum_{i=1}^{m}\beta_i\xi_i\right\}$$

with $\alpha_i, \beta_i \geq 0$.

By finding support vectors which form the hyperplane that can separate instances with the highest margin and least penalty (in a soft margin method), this technique is considered as a linear classifier. However, most datasets in classification problems are not linear separable, so in order to make a support vector machine to effectively work on these datasets, the classifier is needed to be transformed into a non-linear classifier. The technique [10] widely suggested and used is applying the "kernel trick". The kernel trick applies a nonlinear function called a kernel to map instances on the original feature space which might not be linearly separable into the transformed feature space. The transformed feature space is said to be a linear space with an inner product defined. It allows the algorithm to find the better fit maximum-margin hyperplane on this transformed feature space. Examples of the common kernels used for a support vector machine are Gaussian radial basis function [46], polynomial and hyperbolic tangent. A support vector machine has many advantages over other classification models [37]. The algorithm can be applied to various distributions of a dataset due to the inclusion of kernel functions. By formulating the problem into quadratic programming, the solution is given as globally optimal. Moreover, it is robust to outliers by setting the appropriate value of margin parameter. However, applying this algorithm into multiclass classification is still a challenging and debatable problem on how to make it effectively. And since every instance in the training set is used, the data with a large number of instances leads to the large-scaled optimization problem which requires a lot of time and memory to solve and build the model. Moreover, many parameters such as a margin parameter, a kernel function and its parameters are needed to be set appropriately in order to achieve the good performance.

2.1.4 Neural network

A neural network is a machine learning model loosely based on the action of biological neurons. It is formed by a network of weighted and nonlinear transfer functions. One of well-known and mostly used neural network models is multilayer perceptron. A multilayer perceptron is a supervised learning neural network model first introduced by Rumelhart et al [47] in 1986. It is a network of simple neurons called perceptrons. The basic concept of a single perceptron was introduced by Rosenblatt [48] in 1958. The perceptron computes a single output from multiple real-valued inputs by forming a linear combination according to its input weights and then putting the output through some nonlinear activation function. The multilayer perceptron [49] consists of multiple layers of neuron nodes interconnected in a feed-forward direction. Two layers on each side of network are called the input layer and the output layer, while the rest are hidden layers. The input signal is sent through the network in the forward direction layer-by-layer. Multilayer perceptrons have been applied successfully to solve difficult and diverse problems by training them with a popular supervised learning algorithm known as a back-propagation algorithm. It has three distinctive characteristics:

1. The model of each neuron in the network includes a nonlinear activation function which is differentiable on the entire domain. A commonly used nonlinear function that satisfies and is used widely is a sigmoid logistic function: $y_j = \frac{1}{1+e^{-v_j}}$ where v_j is the weighted sum of all synaptic inputs plus the bias of the neuron *j*, and y_j is the output of the neuron. The nonlinearity is important for the algorithm; otherwise the network becomes only a single-layer perceptron.
2. There are at least one or more layers of hidden neurons that are not parts of the input or output layer. These hidden layers enable the network to learn complex tasks by extracting progressively more meaningful features from the input instances.
3. The network has high degrees of connectivity, determined by the synapses of the network. A change in the connectivity of the network requires a change in the population of synaptic connections or their weights.

The combination of these characteristics provides the multilayer perceptron the ability to learn from experience through training. However, it also leads to drawbacks of the multilayer perceptron such as the difficulty on theoretical analysis of algorithm which is caused by its nonlinearity and high connectivity of the network and the use of hidden layer which makes the learning harder to visualize.

The architectural graph of a multilayer perceptron with two hidden layers and an output layer is shown as an example in figure 4. The network shown here is fully connected. This means a neuron in any layer is connected to all nodes/neurons from the previous layer.

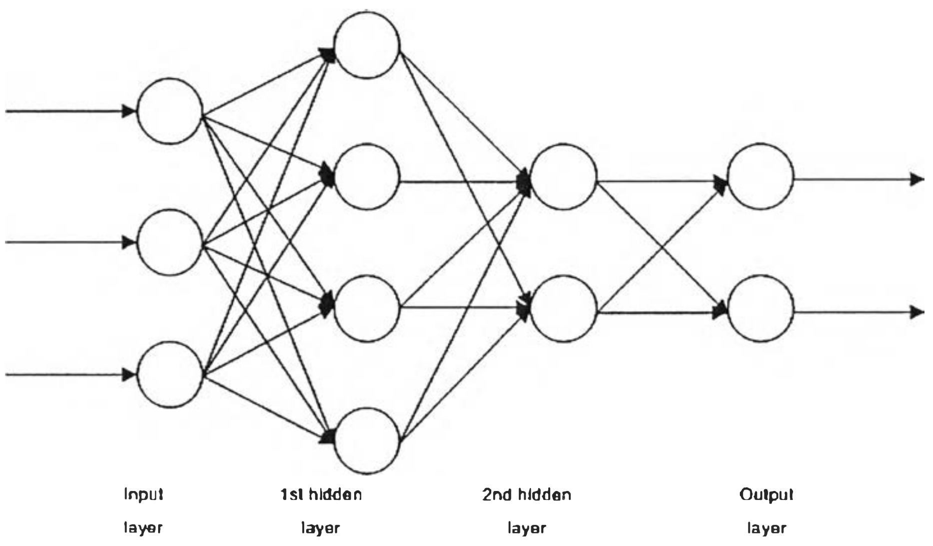


Figure 4: The multilayer in neural network

In the input layer, one neuron represents each attribute of a dataset and has directed connections to every neuron of the subsequent layer. As previously mentioned, the units of these networks apply a sigmoid function as an activation function. Each element from a vector of an instance $X=(x_1,x_2,\dots,x_n)$ is sent to each neuron of the input layer. Each connection from a neuron *i* to another neuron *j* contains the weight w_{ji}, while each neuron *j* in hidden and output node also has the weight w_{j0}. Then an input value v_j and an output value y_j for each neuron *j* can be written as

$$v_j = w_{j0} + \sum_{k \in \text{Pred}(j)} w_{jk} y_k$$

$$y_j = f(v_j)$$

where Pred(j) is a set of neurons which has a connection to neuron j and f is a differentiable nonlinear activation function. With a back-propagation scheme, the input data is repeatedly presented to the network and sent through neurons layer-by-layer. Eventually, it provides the output result which is compared to the desired output to calculate the error. This error is then fed back (back-propagated) to the network and used to adjust weights such that the error decreases with each iteration and the neural model gets closer and closer to produce the desired output.

2.1.5 *K*-nearest neighbor (*K*-NN)

K-nearest neighbor (*K*-NN) is a classification model which classifies the target class of instances based on a majority vote of *k* closest instances in a dataset [11]. It is considered as the simplest learning model with the instance being assigned to the class common amongst its k nearest neighbors.

The algorithm of *k*-NN is described as a lazy learning which does not use the training data points to do any generalization; there is no explicit training phase. This means the training phase is fast and simple as it just collected all attribute values of instances. A lack of generalization means that *k*-nearest neighbor keeps all instances in the training set. More exactly, all instances in the training dataset are needed during the testing phase. This is in contrast to other techniques such as a support vector machine where you can discard instances which are not support vectors. However, the testing phase could be costly consuming in both time and memory. More time might be needed as in the worst case; all data instances might be considered during the decision phase.

K-nearest neighbor assumes that the data is in a metric space which means it must contain a defined metric function to calculate the distance values. Instances in the dataset are supposed to be scalar or possibly even multidimensional vectors. Any distance metric can be applied but Euclidean distance is the commonly used one. Each instance of the training dataset consists of a set of vectors and a class label associated with each vector. In the simplest case, it is either + or – (for positive or negative classes). But *k*-nearest neighbor can work equally well with arbitrary number of classes.

As shown in the figure 5, the important parameter for the k-nearest neighbor algorithm is a number "*k*" which decides how many neighbors (where neighbor is defined based on the defined distance metric) influence the classification. This is usually an odd number for binary classification. If $k = 1$, then the algorithm is simply called 1-NN or the nearest neighbor algorithm. When it is used for classifying an unknown instance, *k*-nearest neighbor classifier uses the majority vote of its *k*-nearest neighbors to determine the class that instance belong to. Originally, the vote from each *k* neighbors is equal but there are some “weighted” nearest neighbor algorithms such as Shephard’s method [50] which put higher weight on votes from closer neighbor instances making more influence on deciding which class an unknown instance is.

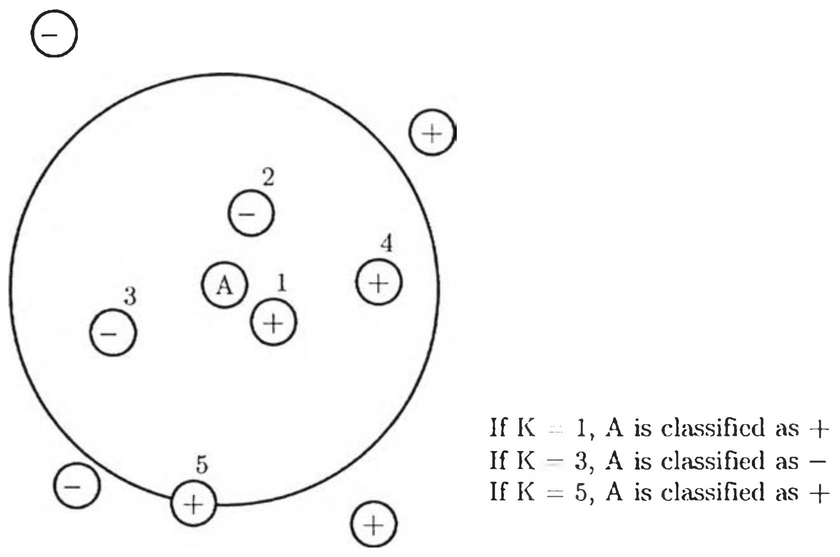

2
4
3
A
1
5
If K = 1, A is classified as +
If K = 3, A is classified as −
If K = 5, A is classified as +

Figure 5: The visualization of *k*-nearest neighbor

As previously mentioned how costly computation time and memory *k*-nearest neighbor has, various data structures are suggested to reduce the time and memory consumptions such as KD-tree [51] or cover tree [52].

There are other effective classifiers in data mining but only these five are used in the experiment. The algorithms of these classifiers treat each instance equally during training classifier in order to maximize overall accuracy, so if one class has an extremely higher number, that class could have a priority to have a maximized accuracy while the accuracy values from other classes might be sacrificed instead. However, if the accuracy from the class with a smaller number of instances is more important for the considered problem, then some additional procedures are needed to deal with this kind of situation. A classification problem with this special situation is called as a class imbalance problem.

2.2 Class imbalance problem

Class imbalance problem [19] focuses on performing classification on the dataset with a high between-class imbalance, where one class severely out-represents another class. Despite that this description can be applied to multiclass dataset, most of class imbalance researches still deal with a binary class problem. The imbalanced dataset can be found in many real-life problems i.e. mammography dataset which is a collection of images gathered from a series of mammography exams performed on a set of patients. This dataset is widely used for testing with various learning algorithms [29] [53] [54] in order to find the pattern or characteristic of images from the patients with cancer. However, the number of non-cancerous patients is far greater than the number of cancer patients. Ideally, a classifier which has a degree of predictive accuracy for both groups is required. In practice, the classifier tends to provide an imbalance degree of accuracy; the majority class has a high prediction rate and the minority class has a low prediction rate. For the problem that the prediction rate of a minority class is far more important, the classifier model as mentioned becomes undesirable and it requires some mechanisms to force classifiers to provide a higher accuracy on the minority class.

When an imbalanced data is used to train a classifier, the patterns or rules which describe the minority class appear less than those of the majority class since minority class instances are outnumbered and underrepresented. In the popular decision tree algorithms [22] [55] [56], the imbalance in datasets affects the splitting criterion at each node of the tree. A decision tree uses a recursive, top-down greedy search algorithm in order to find the best attribute as the split criterion at each node of the tree. By partitioning recursively, the number of minority instances at each node is too small and these instances could be predicted as the majority class. This is an example of the effect of imbalanced dataset which leads to poor predictive accuracy on the minority class in some classifiers. The solutions which are suggested to overcome the imbalance can be categorized into three different strategies [19];

1. Data preprocessing techniques for class imbalance problem.
2. Cost-sensitive learning techniques.
3. Algorithmic techniques for imbalance

Each strategy has different approaches to tackle with a class imbalance problem. The overview of these strategies and examples of techniques in each one are followed in upcoming sections.

2.2.1 Data preprocessing techniques for class imbalance problem.

This group of techniques works on modifying an imbalanced dataset by some mechanisms in order to provide a balanced distribution. Some [57] [58] [59] claim that a balanced distribution gathered by modifying an imbalanced dataset provides the improved overall classification performance compared to one from an imbalanced dataset. One of the simplest ideas is random oversampling which increases the number of minority instances by duplicating randomly selected instances from the existing minority instances and adding them into the training dataset. This provides a mechanism to vary the degree of class distribution to a desired or balanced level. On the other hand, the class distribution can be adjusted in the opposite direction by random undersampling which randomly removes some majority class instances from the original dataset. Despite its simplicity, each method leads to different problems [60] [61] [62] affecting the learning of classifiers. The overfitting issue could occur in a random oversampling technique since values in each attribute of existing minority instances are over-emphasized causing classifiers to overly attach with these values and are inflexible to predict unknown instances whose values in each attribute are not exactly equal to ones of instances in the training set. On the other hand, random undersampling might accidentally remove some important concepts of the majority class. So, additional procedures may be needed to systematically oversampling or undersampling while avoiding these issues. Examples of procedure that are suggested to overcome the loss of information caused by random undersampling methods are EasyEnsemble and BalanceCascade algorithms [63]. For EasyEnsemble, subsets of majority instances are independently and randomly sampled and are used to build classifiers along with a set of minority instances. While in BalanceCascade, a supervised learning approach is used to develop a set of classifiers to select which majority class should be undersampled. The idea of BalanceCascade starts with training the classifier from a set of minority instances and a randomly chosen subset of majority instances. After a given classifier has classified the entire majority instances, the correctly classified instances are removed from a set of majority instances. A set of majority instances to train the new classifier is randomly chosen and combined with a set of minority instances to train the new classifier recursively until the procedure meets the terminated condition.

To avoid the overfitting problem caused by random oversampling with original instances, Chawla [25] suggested synthetic minority oversampling technique (SMOTE) to create new synthetic instances. This technique assumes the similarities in the feature space and generates synthetic instances according to these similarities. By these synthetic instances, a decision region created by training a classifier with this synthetic dataset during the classification process becomes denser and more expanded. Then, the expanded region causes the classifier to recognize more instances as positive. The experiments show that SMOTE gives a high true positive rate which eventually leads to a superior recall than the random oversampling technique or the classification with the original imbalanced dataset. However, this also causes many negative instances to be misclassified. It increases the number of false positive error and decreases the accuracy of the overall classification.

Since SMOTE uses every positive instance to generate synthetic instances without considering its location or neighbors, a synthetic dataset from SMOTE might not represent the actual characteristic of positive instances or might contain too many unnecessary synthetic instances for training classifiers while provides little performance gain. Various oversampling methods have been proposed to overcome this situation such as ADASYN [26], which uses the density distribution as the criterion to decide how many synthetic instances are created from each positive instance, or borderline-SMOTE [27], which uses the number of negative neighbors to classify and select positive instances to be used for generating synthetic instances. Safe-level SMOTE [28] applies the idea of borderline-SMOTE about using the ratio of positive and negative neighbors of each positive instance to determine whether each will be used to generate synthetic instances and define the possible location of these synthetic instances. Another interesting adaption of SMOTE is DBSMOTE [29] which uses the density of positive instances to group them via DBSCAN [14] and creates synthetic instances in each group by assuming the multivariable normal distribution. These techniques provide different characteristics of synthetic datasets and different performance results which could fit with an objective of the class imbalance problem. As follows, these oversampling techniques are used for this dissertation for experiments.

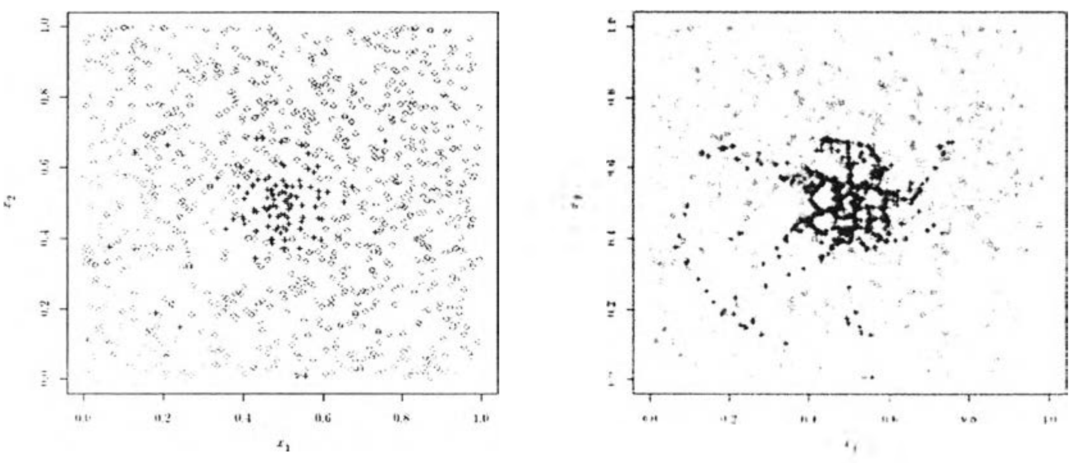
Synthetic oversampling techniques

In this subsection, dealing with the class imbalance problem by creating synthetic minority instances approach is focused. Generating synthetic instances increases the number of positive instances in the imbalanced dataset and the ratio of the number of positive instances and the number of negative instances becomes more balanced. Random oversampling which is also another option to increase the number of minority instances is not considered due to the overfitting problem. There are many oversampling techniques to create synthetic instances which can overcome the overfitting problem but still represent the training dataset well.

2.2.1.1 Synthetic Minority Oversampling TEchnique (SMOTE)

Synthetic Minority Oversampling TEchnique or SMOTE [26] is a synthetic oversampling technique that shows great performances in many applications and implemented in various data-mining tools such as KNIME and Rapidminer. SMOTE assumes the similarities in the feature space and generates the synthetic instances according to these similarities. In SMOTE, each positive instance p finds k-positive nearest neighbors and choose one of them as $\hat{p}$ to form a line segment. A synthetic instance p' is randomly generated on the line segment. Therefore, p' is algebraically written as $p' = p + \lambda(\hat{p} - p)$ where λ is a random number in [0, 1].

The process continues for every positive instance until the number of negative instances and positive instances becomes nearly equal. With these new synthetic instances, a decision region created during the classification process becomes denser and more expanded as shown in figure 6. The expanded region influences a classifier to recognize more instances as positive. The experiments in [26] show that SMOTE gives higher true positive rate which eventually leads to the superior recall value than any other oversampling techniques. However, this also causes many negative instances to be misclassified. It increases false positive error and decreases the accuracy of overall classification.



a) b)

Figure 6: The scatter plots of generated datasets; a) an original imbalanced dataset and b) a balanced dataset with SMOTE

In [26], the decision tree C4.5 [32], Ripper [64] and naïve Bayes classifier [8] are used as classifiers to compare the performance with several undersampling techniques. SMOTE shows the superior AUC value comparing with undersampling schemes set by the author of the paper.

2.2.1.2 Adaptive Synthetic Sampling (ADASYN)

Adaptive Synthetic Sampling or ADASYN is another approach to improve SMOTE introduced by He [26]. In SMOTE, the number of synthetic instances generated by each positive instance is distributed equally to each positive instance. ADASYN proposes using a distribution function to vary the number of synthetic instances generated by each positive instance. The algorithm first finds *k*-nearest neighbors of each positive instance in the entire dataset. The number of negative neighbors of each positive instance is collected and forms a distribution function. The number of synthetic instances created from each positive instance p_i is calculated by the formula

$$r_i = \Delta_i \Big/ \sum_{j=1}^{n} \Delta_j$$

where Δ_i is the number of majority instances that are in *k*-nearest neighbors of i^{th} positive instance and n is the number of positive instances in the dataset. The ratio r_i becomes the ratio of the number of synthetic instances created on this i^{th} positive instance to the total number of required synthetic instances. Since $\sum_{j=1}^{n} \Delta_j$ = 1, this ratio is claimed to be a density distribution. Their paper also claimed that this can improve the recall of classification even better than SMOTE. Figure 7 supports that claim by showing the expanded region of positive instances affected by ADASYN. With this expanded area, the number of prediction as positive is increased by classifiers leading to the increasing number of correctly predicted positive instances and the increasing value of recall. However, this also causes high false positive rate even greater than one from SMOTE. The increasing false positive rate may cause lower F-measure and accuracy values which could be critical for some class imbalance problems.

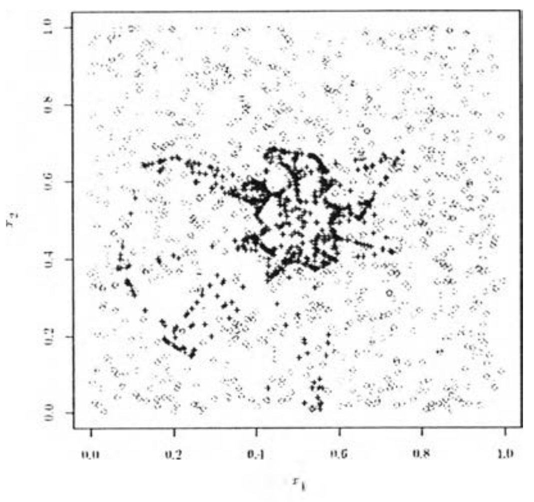


Figure 7: The scatter plot of a generated dataset after balancing with ADASYN

The performance of ADASYN is compared against SMOTE and the original imbalanced dataset with decision tree as a classifier. Under three performance measures; recall, F-measure and geometric mean, the number of datasets which ADASYN can achieve the best value in each performance measure is counted and reported. From the total five datasets, ADASYN get the highest number of datasets with the best value in all three measures. It should be noted that they use only a limited number of datasets and only one classifier.

2.2.1.3 Safe-level SMOTE : Safe-level Synthetic Minority Oversampling TEchnique

Safe-level Synthetic Minority Oversampling TEchnique or safe-level SMOTE is an oversampling technique modified from SMOTE by Bunkhumpornpat et al [28]. Similar to ADASYN, this technique also uses the surrounding or neighbors of each positive instance to indicate how synthetic instances are created. Unlike ADASYN, they do not use the number of neighbors of each positive instance to distribute the number of synthetic instances created, but it is used to define which positive instances should be used to create synthetic instances and to determine the possible location of these synthetic instances. In safe-level SMOTE, the number of positive neighbors for each positive instance is collected and defined as a safe-level value. In order to avoid creating synthetic instances close to negative instances, it generates an instance closer to the positive instance with a higher safe-level value. In their algorithm, the positive instances which have safe-level values equal to zero are considered noise and are dropped from consideration, similar to a concept in borderline-SMOTE [27]. For the rest of the positive instances, their safe-level values take part in a synthetic process to determine the possible location of a synthetic instance. Safe-level SMOTE requires a setting of two parameters, *k* and *c*. The parameter *c* is used exclusively for the safe-level value computing stage as *c*-nearest neighbors of each positive instance are determined and the number of positives in these neighbors is counted for its safe-level value. Then, a positive instance with the non-zero safe-level value, *p*, is used for generating synthetic instances. The instance *p* is paired with one of its *k*-positive nearest neighbors, $\hat{p}$ and the line segment between these two instances is formed. Conveniently, *k* is set to the same value as *c*. In this stage, the safe-level of *p* and $\hat{p}$ computed from the previous stage are used to calculate the safe-level ratio (sl_ratio) which is the proportion of the safe-level value of *p* over the safe-level value of $\hat{p}$. The safe-level values of *p* and $\hat{p}$ along with their safe-level ratio determine the possible range of a synthetic instance per conditions shown in table 1.

Table 1: The condition of safe-level and safe-level ratio and their corresponding ranges

Safe-level of *p* (sl_p)	Safe-level of $\hat{p}$ ($sl_{\hat{p}}$)	Safe-level Ratio (sl_ratio)	The possible range
$sl_p > 0$	$sl_{\hat{p}} = 0$	∞	$[p, p]$
$sl_p > 0$	$sl_{\hat{p}} > 0$	sl_ratio = 1	$[p, \hat{p}]$
$sl_p > 0$	$sl_{\hat{p}} > 0$	sl_ratio > 1	$[p, p + (\hat{p} - p)/\text{sl\_ratio}]$
$sl_p > 0$	$sl_{\hat{p}} > 0$	sl_ratio < 1	$[\hat{p} - \text{sl\_ratio} \cdot (\hat{p} - p), \hat{p}]$

A synthetic instance is created in a random position on the given line segment formed from a pair of positive instances on the given range in table 1. By this modified range, it automatically avoids placing a synthetic instance around the positive instance with a lower safe-level value while a positive instance is synthesized closer to the positive instance with a higher safe-level value as shown in figure 8. The algorithm repeats for all positive instances until it achieves a dataset with the desired balanced number of all classes.

With this concept, safe-level SMOTE indirectly acknowledges the existence of negative instances located around the positive instance in the oversampling process which is different from SMOTE. Since safe-level SMOTE neglects the minority outcasts, one should aware that these instances may be critical in the class imbalance problem and the positive prediction rate may decrease if these instances are excluded. This concern is reflected by the relatively low recall values compared to other oversampling techniques such as SMOTE or ADASYN.

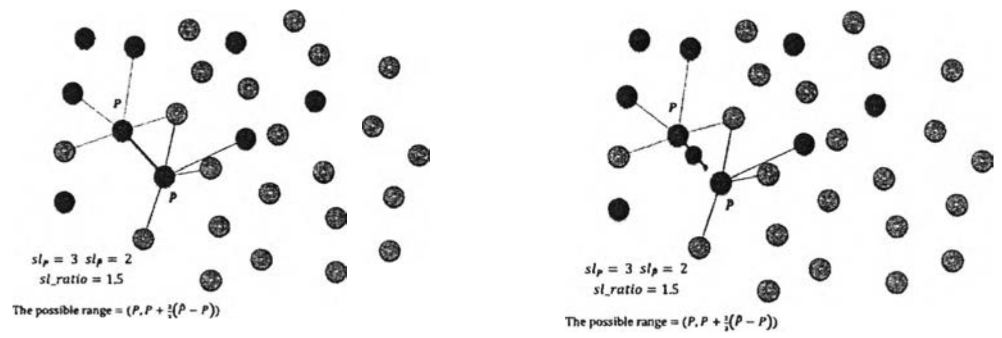


Figure 8: The visualization of SLS on the adjusted range due to the safe-level ratio

The resulting balanced dataset generated by safe-level SMOTE differs from the balanced dataset generated by SMOTE. The significant difference is that there is no synthetic instance created from positive instances that are surrounded heavily by negative instances. Most synthetic instances are placed close to a group of original positive instances as shown in figure 9 which compare the resulting dataset among three different techniques.

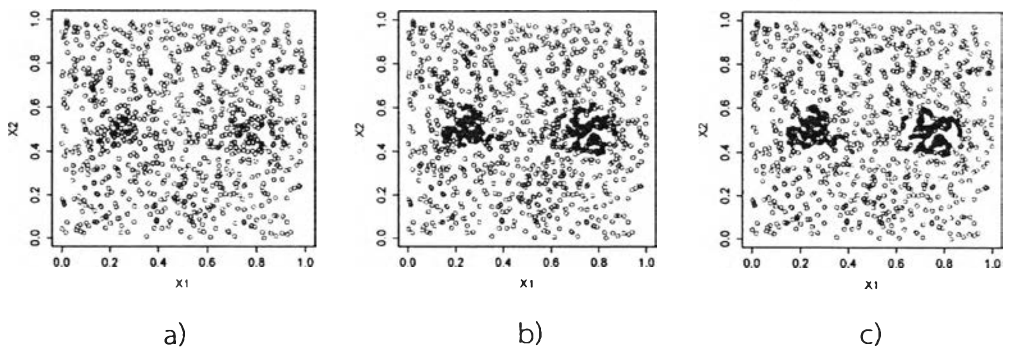


Figure 9: The scatter plots of generated dataset a) an original imbalanced dataset, b) a balanced dataset by SMOTE and c) a balanced dataset by safe-level SMOTE

In [28], the experiment is conducted using three classifiers; decision tree (C4.5) [32], naïve Bayes classifier [8] and support vector machine [9]. Safe-level SMOTE is compared against SMOTE [25] and borderline-SMOTE [27]. However, there are only two UCI repository datasets in this experiment. So it can be debatable whether this algorithm is effective on other real-life datasets.

2.2.1.4 Density-based Synthetic Minority Oversampling TEchnique (DBSMOTE)

Density-based Synthetic Minority Oversampling TEchnique or DBSMOTE [29] is another oversampling technique which aims to create synthetic instances to mimic the multivariate normal distribution. So, the ideal distribution of each set of instances should be dense around the core and sparse around the border. In order to achieve a multivariate normal distribution, DBSCAN [14] which is one of density-based clustering algorithm is used to group minority instances based on their density.

Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

Density-Based Spatial Clustering of Applications with Noise or DBSCAN [14] is a clustering algorithm which groups instances under the defined density. Its concept is originated from human perception about clustering that instances which locate densely and closely together are supposed to be in the same cluster. By given the specific range, *Eps*, and the density threshold, *MinPts*, an instance p is directly density-reachable with an instance q if p is one of *Eps*-neighbor of q and q has the number of *Eps*-neighbor more than *MinPts*. If there is a chain of instances, p_0, p_1,..., p_{n-1}, p_n ; $p_0 = p$, $p_n = q$ which p_i are directly density-reachable with p_{i+1}, then it can be said that p is density-reachable with q. For any two instances which are both density-reachable with the same instance, these two instances are called density-connected. By these definitions, a cluster is formed as instances which are density-connected with respect to the defined *Eps* and *MinPts*, while an instance which does not belong to any cluster is identified as noise.

Unlike some other well-known clustering algorithms, the number of resulting clusters is identified by the algorithm itself and does not need to be assigned from a user. DBSCAN also can detect a group of instances which are non-convex. Moreover, the possible existence of outliers does not affect the outcome of DBSCAN since it is detected as noise. However, DBSCAN also has several major drawbacks such as its two parameters, *Eps* and *MinPts* are needed to be chosen wisely since they can strongly influence the clustering result and DBSCAN is unable to deal with the dataset with varying density.

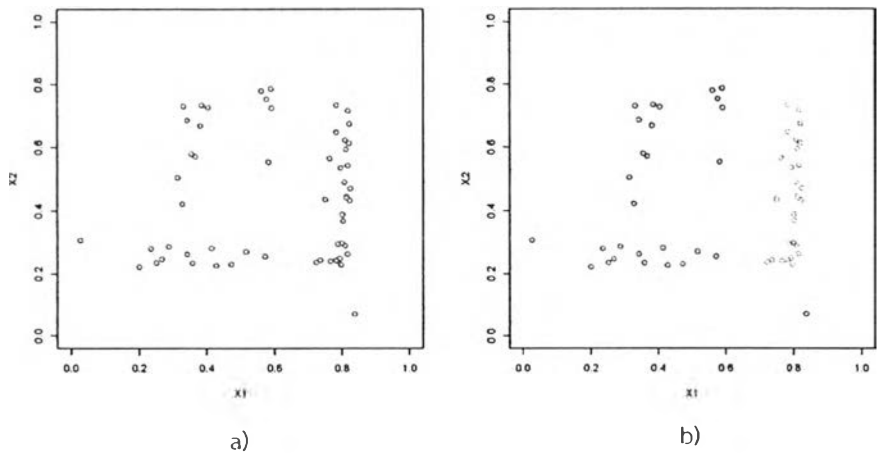


Figure 10: The scatter plots of generated dataset; a) an original imbalanced dataset and b) positive instances clustering with DBSCAN

After performing DBSCAN on positive instances in a dataset, each cluster of positive instances is considered. In each cluster, the graph which contains instances in the cluster as its vertices is formed. The edges of the graph link between two vertices which are directly density reachable to each other and the weight of the edge is the distance value between those two vertices. The pseudo-centroid of the cluster which is an instance closest to the centroid of a cluster is also assigned. Then, a synthetic instance is created on the shortest path from each minority instance (P) to the pseudo-centroid (PC_1) of its cluster. This leads the resulting synthetic dataset to be dense around the core of each group of original minority instances.

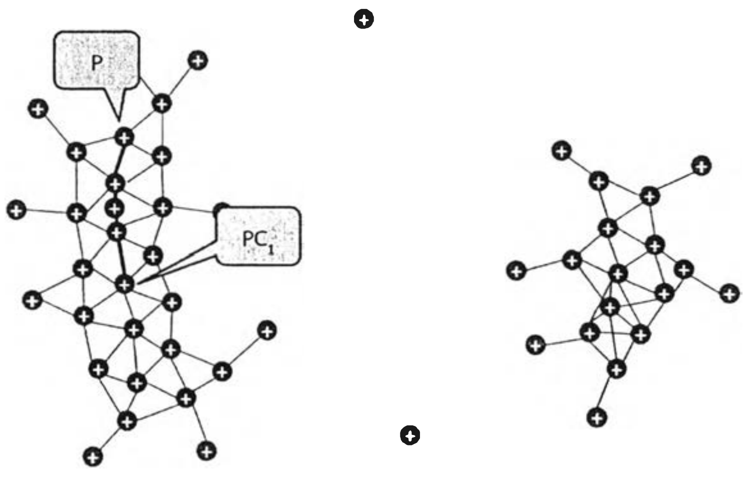


Figure 11: The synthetic generation process of DBSMOTE

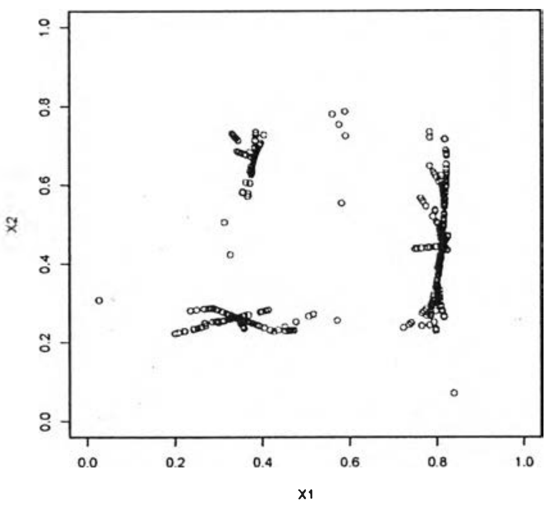


Figure 12: The scatter plot of a generated dataset after balanced by DBSMOTE

Since this oversampling technique is relying heavily on DBSCAN, the advantages and disadvantages of DBSCAN [14] also inherit into DBSMOTE such as DBSMOTE can generate synthetic instances on the non-convex shape which is flexible for any shape of the original minority set, it is insensitive to the existence of outliers but the settings of *Eps* and *MinPts* influence the result of the algorithm.

In [29], seven imbalanced datasets from UCI are used to compare the performance with the original imbalanced dataset, SMOTE [25], borderline-SMOTE [27] and safe-level SMOTE [28]. These oversampling techniques are applied with five classifiers; C4.5 [32], naïve Bayes classifier, support vector machine, Ripper [64] and *k*-nearest neighbor [11]. The accuracy performance of these oversampling techniques is represented and compared on four performance measures; precision, recall, F-measure and AUC. DBSMOTE provides the best results on many cases of datasets and classifiers. The improvement of performance is verified its significance using Student's paired t-test.

These oversampling techniques provide different balanced datasets based on synthetic instances which are differently created. Each of resulting datasets also leads to different classification model despite of being trained from the same classification algorithm. In order to compare the performance from these different models, performance measures which are consistent with the class imbalance problem are needed. Then, some related measures are reviewed.

2.2.2 Cost-sensitive learning techniques.

Cost-sensitive learning [65] is another machine learning which can be applied to deal with class imbalance problem. It considers the misclassification costs in order to minimize the total cost. Generally, most original classification algorithms aim to minimize the error rate: the percentage of the incorrect prediction of class labels. They ignore the difference between types of misclassification errors. In particular, they implicitly assume that all misclassification errors cost equally. However, this assumption is not true in class imbalance problem. For example, in medical diagnosis of a certain cancer, if the cancer is regarded as the positive class, and non-cancer (healthy) as negative, then missing a cancer (the patient is actually positive but is classified as negative; thus it is also called "false negative") is much more serious (thus expensive) than the false-positive error. Various empirical studies [66] [67] have shown that cost-sensitive learning is very effective in a certain domain of imbalanced datasets. So, this group of techniques becomes a suitable option for dealing with class imbalance problems.

The theory of cost-sensitive learning is described by Elkan [68]. The theory describes how the misclassification cost plays its essential role in various cost-sensitive learning algorithms. In cost-sensitive learning for binary classification, the costs of false positive (actual negative but predicted as positive; denoted as FP), false negative (FN), true positive (TP) and true negative (TN) can be given in a cost matrix, as shown in table 2. In the table, the notation C(*i*, *j*) is used to represent the misclassification cost of classifying an instance from its actual class *j* into the predicted class *i*. (+ for positive, and - for negative). These misclassification cost values can be given by domain experts, or learned via other approaches. In cost-sensitive learning, it is usually assume that such a cost matrix is given and known. For multiclass problem, the cost matrix can be easily extended by adding more rows and more columns.

Table 2: A cost matrix of binary classification

		Predicted Class	
		Positive	Negative
Actual Class	Positive	C(+, +)	C(-, +)
	Negative	C(+, -)	C(- , -)

Note that the cost values in the diagonal (TP and TN) is usually regarded as the "benefit" when an instance is predicted correctly. Since this is class imbalance problem where the accuracy of positive class is preferred, it is often more expensive to misclassify an actual positive example into negative than an actual negative example into positive. That is, the value of FN or C(-, +) is usually larger than that of FP or C(+, -).

Given the cost matrix, an example should be classified into the class that has the minimum expected cost. This is the minimum expected cost principle. The conditional risk R(*i* | *x*) of classifying an instance x into class *i* by a classifier can be expressed as:

$$R(i \mid x) = \sum_{j \in \{+,-\}} P(j \mid x) C(i, j)$$

where $P(j \mid x)$ is the probability estimation of classifying an instance into class *j*. That is, the classifier classifies an instance x into the positive class if and only if:

$$P(- \mid x) \cdot C(+, -) + P(+ \mid x) \cdot C(+, +) \leq P(- \mid x) \cdot C(-, -) + P(+ \mid x) \cdot C(-, +)$$

This is equivalent to:

$$P(- \mid x) \cdot (C(+, -) - C(-, -)) \leq P(+ \mid x) \cdot (C(-, +) - C(+, +))$$

Thus, the decision (of classifying an example into positive) is not changed if a constant is added into a column of the original cost matrix. Thus, the original cost matrix can always be converted to a simpler one by subtracting C(-, -) to the first column, and C(+, +) to the second column. After such conversion, the simpler cost matrix is shown in table 2. Thus, any given cost-matrix can be converted to one with C(- , -) = C(+, +) = 0. The classifier classifies an instance *x* into positive class if and only if:

$$P(- \mid x) \cdot C(+, -) \leq P(+ \mid x) \cdot C(-, +)$$

Table 3: A simpler cost matrix with an equivalent optimal classification.

		Predicted Class	
		Positive	Negative
Actual Class	Positive	0	C(-, +) - C(+, +)
	Negative	C(+, -) - C(- , -)	0

As P(- | x) = 1 – P(+ | x), A threshold p^* can be obtained for the classifier to classify an instance x into positive if P(+| x) ≥ p^*, where

$$p^* = \frac{C(+,-)}{C(+,-) + C(-,+)}$$

Thus, if a cost-insensitive classifier can produce a posterior probability estimation p(+ | x) for an instance x in the test set, it can be transformed into cost-sensitive by simply choosing the classification, and classifying any instances to be positive whenever P(+ |x) ≥ p^*. This approach is used on several cost-sensitive meta-learning algorithms, such as Relabeling. However, some cost-insensitive classifiers, such as C4.5, may not be able to produce accurate probability estimation; they are designed to predict the class correctly. Empirical thresholding [69] does not require accurate estimation of probabilities – an accurate ranking is sufficient. It simply uses cross-validation to search the best probability from the training instances as the threshold. Traditional cost-insensitive classifiers are designed to predict the class in terms of a default, fixed threshold of 0.5. Then, the original training set instances are rebalanced by sampling such that the classifiers with the 0.5 threshold is equivalent to the classifiers with the p^* threshold, in order to achieve cost-sensitivity. The rebalance is done as follows. If all positive examples are kept and assumed as the rare class, then the number of negative examples should be multiplied by the cost ratio C(+, -)/C(- , +) = FP/FN. Note that as usually FP < FN, this ratio is less than 1. This is thus often called "under-sampling the majority class". This is also equivalent to "proportional sampling", where positive and negative examples are sampled by the ratio of p(+) : p(-) where p(+) and p(-) are the prior probabilities of positive and negative instances in the original training set. That is, the prior probabilities and the costs are interchangeable: doubling p(+) has the same effect as doubling false negative, or halving false positive [70].

Cost-sensitive learning can be categorized into two categories. The first one is the direct method which designs a classifier which contains the cost-sensitive approach. An example of direct methods is ICET [71] which incorporates misclassification cost in the fitness function of the genetic algorithm. Another method in this group is a cost-sensitive decision tree (CSTree) [72] which uses the misclassification costs directly in its tree building process. In place of minimizing entropy in attribute selection, CSTree chooses the best attribute by the expected total cost reduction. So the resulting tree is built under the objective that it minimizes the total misclassification cost. The idea of replacing the objective function of each classifier with the cost minimization function is also applied with other classifiers such as neural network or support vector machine.

The other one is the meta-learning method which provides the components which converts a cost-insensitive classifier into the cost-sensitive classifier. One approach in this group is applying cost items or cost adjustment functions on the weight updating process in the boosting scheme such as AdaBoost [73] and AdaCost [74]. Some methods such as MetaCost [75] or CSC [76] apply the threshold p^* to classify instances if that cost-insensitive classifier can produce probability estimations. Another method which uses this threshold is a cost-sensitive naïve Bayes [77] which uses it to classify instances based on the computed posterior probability. Similarly, some sampling meta-learning methods, such as Costing [78], are also used for class imbalance problem using the cost ratio to help classifying, instead.

The advantage of the cost-sensitive learning is that it can be adapted into various classifiers. However, the way to apply a cost-sensitive approach depends on the type of classifiers. Except for cost-sensitive algorithms which include some sampling ideas, most methods do not alter the size of the original dataset. It is shown that it works well with the large size of imbalanced dataset (more than 10,000 instances); however, repetitive classification runs may be required in order to find the optimal cost which need time and memory resources.

2.2.3 Algorithmic techniques for class imbalance problem

Another approach to deal with class imbalance problem is modifying the existing classification algorithms to deal with an imbalanced dataset. Techniques in this group are kernel modification methods for imbalanced learning, the integration of kernel methods with sampling methods and active learning techniques for imbalanced learning. Kernel is an additional function added in support vector machine to transform the space of a dataset into the one which support vector machine can be effectively classified instances. The imbalanced dataset affects the performance of a support vector machine strongly. Since one of the objectives of support vector machine is to minimize the total error, it is biased toward the majority class. In most cases, a binary class space is separated by the neighborhood of the majority class. It is possible that support vectors representing the minority class are located away from the ideal line and contribute less to the classification result. In order to shift this situation to another space where the separation between these two classes becomes achievable, the kernel concept is needed.

An example of techniques which modifies the kernel to respond with imbalanced datasets is the kernel classifier construction algorithm based on the orthogonal forward selection and the regularized orthogonal weighted least squares estimator (ROWLS) [79]. It integrates the concept of leave-one-out cross validation and the area under curve evaluation metric to develop an LOO-AUC objective function to find the most optimal kernel model. The high weight is assigned to the misclassified instances in minority class via the cost sensitivity of the parameter estimation cost function in the ROWLS algorithm. Another example is a group of techniques which adjusts the SVM class boundary. Wu and Chang [80] suggest three approaches for adjusting boundary skews; the boundary movement (BM) approach, the biased penalties (BPs) approach and the class-boundary alignment (CBA). Another method in this group is the kernel-boundary alignment (KBA) algorithm [81] which is to modify the kernel matrix generated by a kernel function according to the imbalanced data distribution. This algorithm applies the adaptive conformal transformation (ACT) methodology [82] where the conformal transformation on a kernel function is based on the feature-space distance and the class imbalance ratio. To improve SVM robustness, the total margin-based adaptive fuzzy SVM kernel method (TAF-SVM) [83] is introduced. Another interesting kernel modification method for imbalance learning is the *k*-category proximal support vector machine with Newton refinement [84] which tries to transform the soft-margin maximization paradigm into a simple system of *k* linear equations, where *k* is the number of classes. By doing this, this technique has a very fast learning procedure since it requires only solving a system of linear equations. Another technique suggested by Raskutti and Kowalcyzk [78] is to compensate the weight of sampling and data space when one of the classes is ignored in SVM. It can be noted that many kernel-based learning methods use the hybrid approach of sampling and ensemble techniques along with kernel modification methods to improve the performance.

Another direction of this group of techniques dealing with imbalance is active learning. SVM-based active learning aims to select the most informative instances from unseen training data in order to retrain the kernel based model. The method proposed by Ertekin et al [85] queries a small subset of data at each iterative step instead of the entire data. Once a SVM is trained from a small subset of the training set, the most informative instances are extracted and form a new training set according to the developed hyperplane. Finally, a new training set and all unseen training instances are used to actively rebuild the SVM using the LASVM online SVM learning algorithm [86]. However, the distance recalculation between each instance and the current hyperplane cause the search process for the most informative instances becomes computationally expensive. To fix it, the method to effectively select informative instances from a random subset of training set is suggested to reduce the computational cost.

Even this group contains methods which are effective, applicable with various imbalanced dataset and widely accepted from many researchers, most methods in this group are specific to the support vector machine. Therefore, if support vector machine is not suitable for the domain of these datasets, most techniques from this group may not be practical for the problem.

Despite the techniques for dealing with a class imbalance problem are categorized into three groups, the ensemble between techniques from different groups are frequently suggested by many researchers. Those include SMOTE with Different Costs methods and several over/undersampled SVMs which combines sampling approaches with another algorithmic approach to achieve a new algorithm effectively deal with class imbalance problem.

This dissertation concentrates on oversampling techniques to balance a class distribution of datasets. These techniques have several advantages such as they can be adapted to a general classifier since they only modify the original dataset. The resulting dataset becomes a balanced dataset which has no problem when using a general classifier. So this group of techniques can be effectively applied to various classifiers.

In order to measure the performance, it can be assessed through performance measures. In the next section, a collection of performance measures used in this dissertation is introduced. Then, only some measures which are suitable to an imbalance problem in this dissertation are focused.

2.3 Performance measures

Performance measures are used to determine the effectiveness of each classifier in classification. Since this dissertation focuses on a binary classification, the basic terms and descriptions can be seen in 2x2 confusion matrix as below.

Table 4: A confusion matrix of binary classification

		Predicted Class	
		Positive	Negative
Actual Class	Positive	True Positive	False Negative
	Negative	False Positive	True Negative

As shown in the confusion matrix [87], there are 4 types of result for classified instances. True positive is the number of positive instances correctly classified as positive. True negative is the number of negative instances correctly classified as negative. False negative is the number of positive instances incorrectly classified as negative and finally false positive is the number of negative instances incorrectly classified as positive. Then, several values are defined such as a true positive rate which is the ratio of true positive over the total number of instances identified as positive, a false positive rate which is the ratio of false positive over the total number of instances identified as positive, a true negative rate which is the ratio of true negative over the total number of instances identified as negative and a false negative rate which is the ratio of false negative over the total number of instances identified as negative.

Accuracy is the simplest performance measure which is often used in general classification problem. It is the total number of correctly recognized instances divided by the total number of instances. This measure is not suitable for the class imbalance problem which focuses on the prediction of the minority class. The model with the high true negative rate but low true positive rate could get a high accuracy despite being the unsuitable model for a class imbalance problem.

Due to the nature of class imbalance problem, a classification model training from the original dataset has a high accuracy since it predicts instances which are mostly majority class correctly. A model which is trained through a synthetic dataset from the sampling technique sacrifices the correctness of predicting majority class for the higher minority prediction rate. Therefore, the model from the original dataset usually achieves higher accuracy than one from a synthetic dataset. This is a reason the accuracy is unsuitable performance measure for class imbalance problem.

Precision is calculated by the true positive divided by the sum of true positive and false positive. It reflects the reliability of classifier in classifying any instances as positive. By its definition, there are two factors which contribute the high precision value; the high true positive value which is desirable for any classification models, especially in a class imbalance problem and the low number of instances classified as positive. The model from the original dataset usually provides the low number of instances classified as positive, so its precision value is normally higher than the precision of the model from the synthetic dataset. This effect suggests that this measure might not be suitable to be used alone for the class imbalance problem. However, the model with the high precision also has the low false positive rate which may be important in some class imbalance problems.

Recall is determined by the ratio of true positive to the total number of positive instances, i.e. the sum of true positive and false negative. This is suitable with the objective of the class imbalance problem which needs the model that can correctly classify positive instances as much as possible. However, a high recall can be achieved by the model which predicts every instance as positive which gives zero value for the true negative. Therefore, only the recall value is not enough to determine a suitable model for the class imbalance problem. SMOTE [25] is one of the sampling techniques that provide very high recall value. It generates synthetic positive instances from every original positive instance. These synthetic instances expand the decision region for positive regardless of surrounding negative instances. The expanded region could lead the model to have a high false positive rate.

So in order to determine the best classification model in the class imbalance problem, the performance measure that balances between precision and recall is required. This leads to F-measure, geometric mean and adjusted G-mean. F-measure is calculated with the following formula.

$$Fmeasure_{\beta} = \frac{(1+\beta^2) \times precision \times recall}{\beta^2 \times precision + recall}$$

where non-negative real β is the weight users put to emphasize the importance of recall over the importance of precision. F-measure is the harmonic mean of precision and recall when a parameter β is set to 1. The model with a high F-measure implies that it has both high precision and recall.

Geometric mean [88] in machine learning is the square root of the product of sensitivity (recall) and specificity (true negative rate). In order to achieve high geometric mean, the model should have the high number of correctly classified instances in both positive and negative class. The other measure to be considered is adjusted g-mean (AG) [31] whose formula includes the percentage of majority class. It is claimed to detect the model that has the high true positive rate while sacrifices the lower negative prediction rate. The formula of adjusted g-mean is

$$AG = \begin{cases} \dfrac{\sqrt{SP \times SE} + SP \times N_n}{1 + N_n}; & SE > 0 \\ 0; & SE = 0 \end{cases}$$

when *SE* is sensitivity, *SP* is specificity and N_n is the percentage of majority (negative) class. This formula of an adjusted g-mean causes its value to be more sensitive to the change of specificity. The model with a high adjusted g-mean value provides a high classification prediction rate on both classes which could serve a purpose of some class imbalance problems such as bioinformatics or medical datasets.

Chapter 2 has provided the detail of each classifier using in this dissertation, class imbalance problem and existing strategies to deal with the problem. But only the strategy of performing data preprocessing technique such as oversampling techniques is specified here as the main approach to deal with class imbalance problem. Then, some oversampling techniques are introduced since they are used to compare with suggested techniques. Moreover, performance measures are brought up to explain why a certain measure such as F-measure is used in the result. In the next chapter, each issue of oversampling techniques and dissertation concepts are discussed.