CHAPTER 3

MINORITY OVERSAMPLING FRAMEWORK FOR CLASS IMBALANCE PROBLEM

As mentioned from the previous chapter, dealing with an imbalanced dataset by creating synthetic minority instances to balance class distribution of the dataset is an effective approach. However, current techniques have some limitations and weaknesses. In this chapter, the framework which explores those limitations and weaknesses and new oversampling techniques modified from some existing oversampling algorithms based on this framework are introduced.

3.1 Minority outcast handling

The first problem of SMOTE is how to utilize positive instances which are placed away from other positive instances. Normally, these positive instances are viewed as insignificant instances since its number is very low comparing to the number of instances in the entire dataset, especially in class imbalance problem where the total number of positive instances is already low. Some classifiers such as decision tree or support vector machine prefer misclassifying them as negative for the sake of overall accuracy performance.

Borderline-SMOTE [27] and safe-level SMOTE [28] define positive instances whose *c*-nearest neighbors are all negative as danger/noise instances and exclude them for being used to generate synthetic instances. DBSMOTE [29] also identifies positive instances which do not belong to any density-based clusters as noise and excludes them. On the other hand, ADASYN [26] considers these noises whose definition are the same as ones from borderline-SMOTE [27] and safe-level SMOTE [28] as critical to have a number of additional synthetic instances generated around these instances in order to make a classifier recognize them as positive. Since borderline-SMOTE, safe-level SMOTE and ADASYN all use the criterion to determine that positive instances whose *c* nearest neighbors are all negative are special instances that require extra treatments, this dissertation also adopts the same criterion and renames positive instances under this criterion as **minority outcasts**. An example of a minority outcast in a dataset is illustrated in figure 13. On the side note, the original SMOTE [25] treats these noises or minority outcast instances as normal positive instances. In a process of creating synthetic instances from an imbalanced dataset containing these minority outcasts, synthetic instances generated from these minority outcasts usually expand the decision region in a negative region which

locates between these outcasts and their positive neighbors. This expanded decision region ultimately increases the number of positive predictions and causes the increase of recall since there are more actual positive instances predicted correctly. However, this also increases the number of false positive instances as the trade-off for increasing recall. In some class imbalance problems where a false positive error is crucial, a misleading positive region caused by synthetic instances generated from these minority outcasts might not be appropriated. As mentioned prior, some algorithms such as borderline-SMOTE [27], safe-level SMOTE [28] and DBSMOTE [29] consider them as noises and exclude them from being used to generate synthetic instances. This idea can lower the false positive error. However, since every positive instance is critical and important to represent the positive class in the class imbalance problem, completely neglecting these instances may not be appropriate. An alternative approach should be considered.

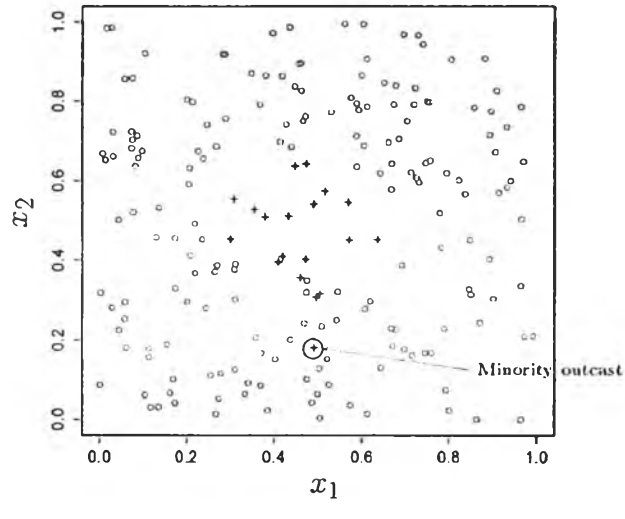

Figure 13: An example of a minority outcast in a dataset

In this section, an alternative technique which specifically deals with these minority outcasts is introduced. It starts with identifying which positive instance is a minority outcast using *c*-nearest neighbors. After minority outcast instances are identified and removed, the rest of positive instances are used to generate synthetic instances based on a selected oversampling technique in order to obtain a balanced dataset. The minority outcasts which are removed in the previous stage are added to a set of negative instances for a new training set. This new training set is used to train a 1-nearest neighbor model. To utilize this model, unknown instances from a test set

are sent to this 1-nearest neighbor model (1-NN model) in order to refine their classification results after the original classification on these instances is finished. If any unknown instances are classified as positive by this 1-NN model, they are automatically classified as positive regardless of the result from the trained classifier. This additional process is called a **minority outcast handling process with 1-NN or minority outcast handling.** It is an extension to other oversampling techniques featuring an outcast exclusion process. It captures missing positive regions caused by a sampling process or a potentially important positive instance which is hardly found due to a small number of positive instances. The algorithm for extracting outcasts is given in algorithm 1 and the algorithm for the minority outcast handling is given in algorithm 2.

Algorithm 1: Minority Outcast extraction algorithm

Data: A numerical-attribute binary class dataset D containing a set of positive instances P and a set of negative instances N and the value of c.

Result: A set of positive instances which are not outcasts P', a set of minority outcasts OC, a vector of safe-level values of P', SL (for SLS), a vector of negative nearest distance E_{oc} (for ANS) and a vector of positive nearest distance $E_{P'}$ (for ANS)

1. OutcastExtraction(D, P, c)
2. For $p_i \in P$ do
3. Identify all c-nearest neighbor of p_i in D
4. Count the number of neighbors of p_i which are positive as sl_i (safe level) of p_i
5. If $sl_i = 0$ then
6. Place p_i in OC
7. Keep the distance between p_i and its nearest negative neighbor as ω_i in E_{oc}
8. End if
9. Otherwise, Place p_i in P' and collect sl_i in SL
10. Keep the distance between p_i and its nearest positive neighbor as ε_i in $E_{P'}$
11. Return { P', OC, SL, E}

Algorithm 2: Minority outcast handling algorithm

Data: A numerical-attribute binary class dataset D containing a set of negative instances N, a set of minority outcast instances OC and a set of unknown instances U.
Result: A vector of assigned class CL (cl_1, cl_2, ..., cl_i) of U.

1. OutcastHandling(D, N, OC, U)
2. For $u_i \in U$ do
3. Calculate the distance from u_i to every instance in a set N and OC
4. Let u^* as argmin{$d(u_i, x) | x \in N$ or $x \in OC$}
5. If $u^* \in OC$ then
6. $cl_i = +$
7. End if
8. Otherwise, $cl_i = -$
9. Return CL

In the minority outcast extraction process, a parameter c is related to the number of minority outcasts since it is the number of nearest neighbors which are used to decide the outcast. The higher value of the parameter c is, the fewer number of positive instances which have all c-nearest neighbors as negative is. If c is set to 1, it could turn the whole classification model to a 1-nearest neighbor classifier since many positive instances are defined as minority outcasts and applied with 1-nearest neighbor except ones inside a dense cluster of positive instances.

If c is set too high, the number of outcasts might be too few and minority outcast handling has little effect on the classification result. For the choice of selecting an appropriate c, the amount of noises of positive instances expected to have in each dataset should be considered. It is possible to use some criteria that can distinguish noises such as an outlier scoring algorithm to replace c-nearest neighbor. For this dissertation, the empirical experiment is conducted to help determining the parameter c. This procedure is explained in chapter 4.

In this dissertation, this minority outcast handling is applied into two new oversampling techniques. The empirical results from this process are shown along with the performance of these two new oversampling techniques in the next chapter.

3.2 Triangular minority oversampling technique

The pattern of generating synthetic instance in SMOTE [25] and its variances is to generate a synthetic instance along the line segment between two original

minority instances which is the simplest relationship between these two minority instances. By using a line segment, it requires only one additional parameter and a simple arithmetic calculation in order to create a new synthetic instance. This synthetic instance is guaranteed to be inside the convex set of two positive instances as shown in figure 14. It also helps strengthen the border of minority instance regions to be denser and easier to be detected by a classification algorithm that works on identifying the border between two classes such as a support vector machine [10] or a decision tree [32].

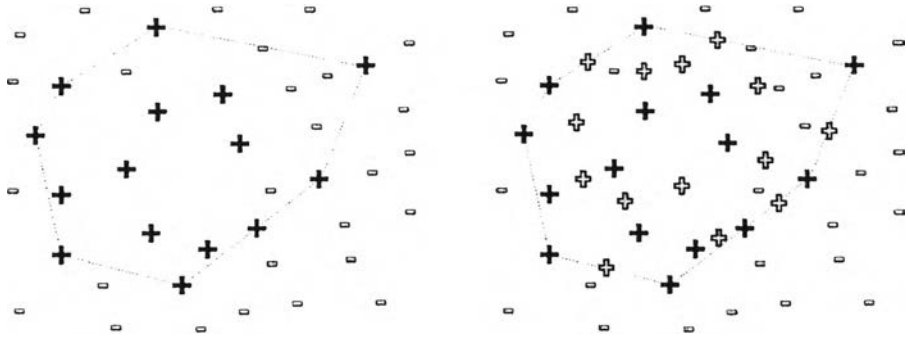

Figure 14: A visualization showing that synthetic instances are not generated outside the convex hull of the original positive region

One question is raised as what would happen if synthetic instances are generated inside the area forming by three or more minority instances instead of the line segment between two. Since oversampling techniques generally aim to increase the density of this region, synthetic instances which are created in the convex hull of three or more minority instances are reasonable. To investigate whether this approach is actually effective, an oversampling technique that employs this idea is designed and the experiments with benchmark datasets are conducted.

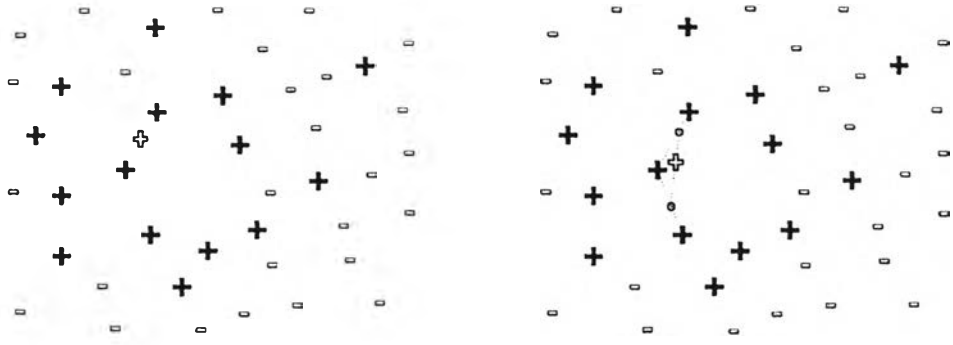

Figure 15: The comparison of synthetic generation of SMOTE and TMOT

The designed oversampling technique is called Triangular minority oversampling technique (TMOT). For each minority instance, it forms two line segments to two of its positive nearest neighbors and creates two pseudo end points on these two lines. Then, a new synthetic instance is generated on a line segment between these two pseudo end points, causing the new instance to be inside the triangle formed by edges linking a minority instance and its two nearest neighbors. It is trivial to see that this technique does not create synthetic instances on the border of a set of minority instances. This result is similar to synthetic instances generated with DBSMOTE [29] where synthetic instances are generated only on the path to the pseudo centroid of the cluster of positive instances, not around the border of the cluster. Later in chapter 4, the classification result from this technique is compared using the original dataset and SMOTE algorithm.

3.3 Relocating framework for safe-level SMOTE

The common procedure of SMOTE [25] on generating synthetic instances is to randomly create synthetic instance on a line segment between one positive instance and one of its *k*-positive nearest neighbors. This part ignores the existence of surrounding negative instances which might be located around each positive instance. Sometimes, there may be negative instances lying between those positive instances and the synthetic instance is generated among these negative instances. Some oversampling techniques use the existence of surrounding negative instances to configure how synthetic instances are generated. Safe-level SMOTE [28] is one variation of SMOTE that takes account of majority instances around each positive instance to determine the possible position of synthetic instances. Safe-level SMOTE uses the number of positive instances in *c*-nearest neighbor of each positive instance to define its "safe-level" value. Then, the safe-level ratio of two positive instances

that form the line segment is used to define the possible interval that a synthetic instance is created. This interval guarantees that the synthetic instance is not created close to the positive one with lower safe-level value and it is hypothetically able to avoid creating these synthetic instances close to negative instances.

Despite of adapting the safe-level approach to control the possible location of synthetic instances, sometimes synthetic instances are generated too close to existing negative instances. In order to remedy this, **relocating safe-level SMOTE** is suggested which implements an additional step to relocate synthetic instances once they are located too closed to negative instances. This algorithm follows the process of safe-level SMOTE, starting with finding *c*-nearest neighbors of every positive instance and calculating its safe-level value. Then, for each positive instance *p*, one of its k-positive neighbors $\hat{p}$ is selected to form a pair. The safe-level values of these two instances and the safe-level ratio are used to set up the interval on the line segment for synthesizing as shown in table 1. After the location is set randomly on that interval, a supposedly new synthetic instance *p'* is generated. However, this synthetic instance is required to enter the trial whether it is too close to surrounding negative instances. If it is closer to surrounding negative instances than two positive instances which generate it, then the location of this synthetic instance is going to be changed. The direction to move *p'* is going toward the positive instance with a higher safe-level value. The procedure for this trial is set based on the value of the safe-level ratio of the pair that generates *p'*. If the ratio is higher than 1, it implies that *p* is the positive instance with a higher safe-level value. The distance values between the new synthetic instance and surrounding negative instances (which are selected from negative neighbors of *p* and a positive neighbor $\hat{p}$) is compared with the distance value from *p'* to the positive instance *p*. If the distance is larger, a new location of *p'* is needed to be chosen on the line segment between old *p'* and *p*. Similarly, if the safe-level ratio is less than 1, $\hat{p}$ is the positive instance with a higher safe-level value. The distance value from *p'* to the positive instance $\hat{p}$ is used to compare with the distance values between *p'* and surrounding negative instances. If the former is larger, a new location of *p'* is needed to be chosen on the line segment between old *p'* and $\hat{p}$. This process continues until *p'* is closer to positives than negatives. The visualization of this procedure is shown in figure 16.

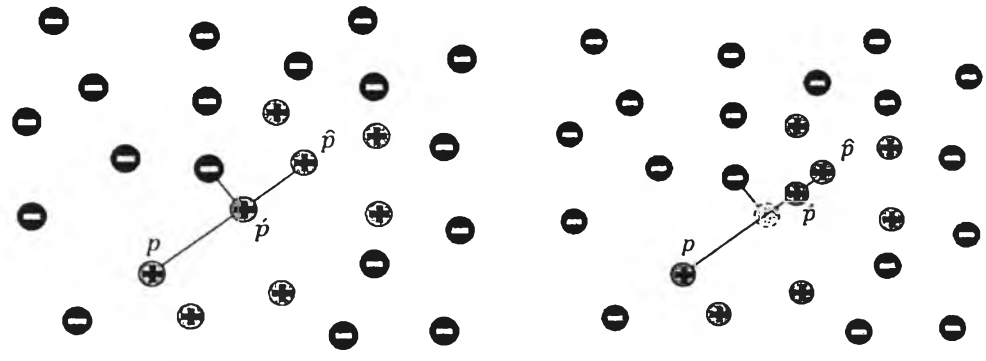

Figure 16: An example of relocating a synthetic instance

This approach is called **relocating safe-level SMOTE (RSLS)**. The algorithm of RSLS is shown in algorithm 3. Apart from the relocating process, this algorithm also includes the minority outcast handling process from section 3.1 to improve the classification performance from the synthetic dataset which is generated from RSLS.

Algorithm 3: Relocating safe-level SMOTE

Data: A numerical-attribute binary class dataset D containing a set of positive instances P and a set of negative instances N, the number of positive nearest neighbor k, and the parameter c.
Result: A nearly balanced dataset which is a combination of D and a set of synthetic instances S, a set of minority outcast OC.

1. Initialization $t = 1$;
2. Find c-nearest neighbor of P in D, let Nc_i be the set of c-nearest neighbors of each $p_i \in P$.
3. Find k-nearest neighbor of P in P, let Pk_i be the set of k-(positive) nearest neighbors of each $p_i \in P$.
4. {*Pused*, *OC*, *SL*, *E*} = OutcastExtraction(*D*, *P*, *c*)
5. While $t < \frac{|N|}{|Pused|}$ do
6. For $p_i \in Pused$ do
7. Random select $\tilde{p}_i$ from PK_i, let sl_j = a safe level of $\tilde{p}_i$ and $safe\_ratio = sl_i/sl_j$
8. gap = GapCalculate(sl_j, $safe\_ratio$)
9. $p' = p_i + gap \times (\tilde{p}_i - p_i)$
10. Let NR be { $nc_i \in NC_i$ | nc_i is negative } $\cup$ { $nc_j \in NC$ of $\tilde{p}_i$ | nc_j is negative }
11. If NR is not empty then
12. If $safe\_ratio \geq 1$ then p' = RelocatingProcess(p_i, p', $safe\_ratio$, NR)
13. If $safe\_ratio < 1$ then p' = RelocatingProcess($\tilde{p}_i$, p', $safe\_ratio$, NR)
14. Add p' into S
15. End for
16. $t = t + 1$
17. End while

1. GapCalculate(sl_i, $safe\_ratio$)
2. If $sl_i > 0$ then
3. If $safe\_ratio = 1$ then set gap as a random number between 0 to 1
4. If $safe\_ratio > 1$ then set gap as a random number between 0 to $1/safe\_ratio$
5. If $safe\_ratio < 1$ then set gap as a random number between $1-safe\_ratio$ to 1
6. If $sl_i = 0$ then set gap as 0
7. Return gap

```
1. RelocatingProcess(p*, p', safe_ratio, NR)
2.     Calculate the distance from p* to p' as d(p*, p' ).
3.     While d(p*, p' ) > min{d(p' , nr)| nr ∈ NR} do
4.         gap = GapCalculate(1, 1)
5.         p' = p* + gap × (p' - p*)
6.     Endwhile
7.     Return p'
```

After RSLS provides a set of synthetic instances S, it is added into the original imbalanced dataset to turn it into the balanced dataset. This balanced dataset is used to train the classifier. Since the number of instances between two classes is no longer imbalanced, the training algorithm can be applied. This could help classifier provide the better prediction rate on minority class instances. Moreover, the way the location of each synthetic instance is controlled, added synthetic instances can better represent minority class and achieve a higher positive prediction rate than other oversampling techniques. Additionally, minority outcasts which are extracted from a set of positive instances are used in the minority outcast handling process introduced in section 3.1. 1-nearest neighbor model which is developed through this process is going to be used after instances in the test set are classified with a trained classifier. The flowchart of the process is shown in figure 17.

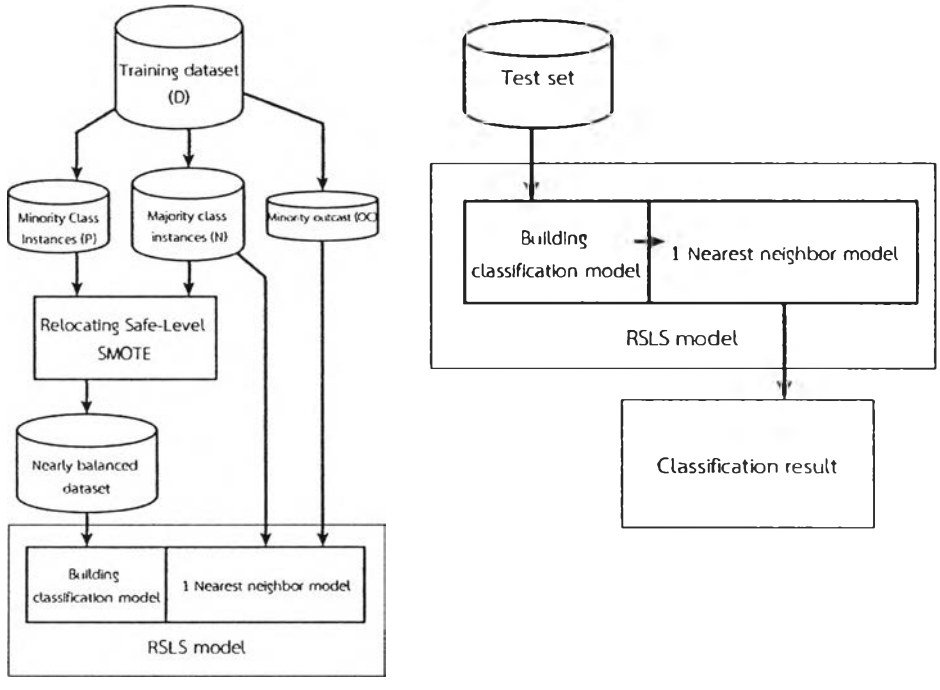

Figure 17: A diagram of relocating safe-level SMOTE with 1-NN minority outcast handling

3.4 Adaptive neighbors Synthetic Minority Oversampling TEchnique under 1-NN outcast handling

Another problem in SMOTE [25] is how to select the appropriate value of *k*, the number of positive neighbors. This parameter determines the number of possible positive neighbors that is chosen to pair up with each positive instance in order to create synthetic instances along the line segment of that pair. In original SMOTE paper [25], Chawla used the value of *k* as 5 for his experiments and this number is inherited to other related oversampling techniques. To verify whether the value of *k* as 5 is actually the optimal value, researchers need to perform various experimental runs.

For this section of the dissertation, a new oversampling technique is introduced under the name "Adaptive Neighbors Synthetic Minority Oversampling Technique" or ANS. This oversampling technique automatically configures the value of *k* used for each positive instance based on the density of surrounding positive instances.

The first step is to exclude minority outcasts from an imbalanced dataset. Positive instances which are not identified as minority outcasts are used for

generating synthetic instances via SMOTE algorithm. Each positive instance requires at least one positive neighbor to form the line segment that generates a synthetic instance. To guarantee that each positive instance p_i contains at least one neighbor, the maximum distance value between pairs of two closest positive neighbors is chosen as the radius. With this value, every positive instance contains at least one positive nearest neighbor under this radius. The number of positive nearest neighbor of p_i is used as the parameter k_i for p_i to generate synthetic instances with SMOTE. The variation of these numbers of k depends on the density of positive instances around each instance.

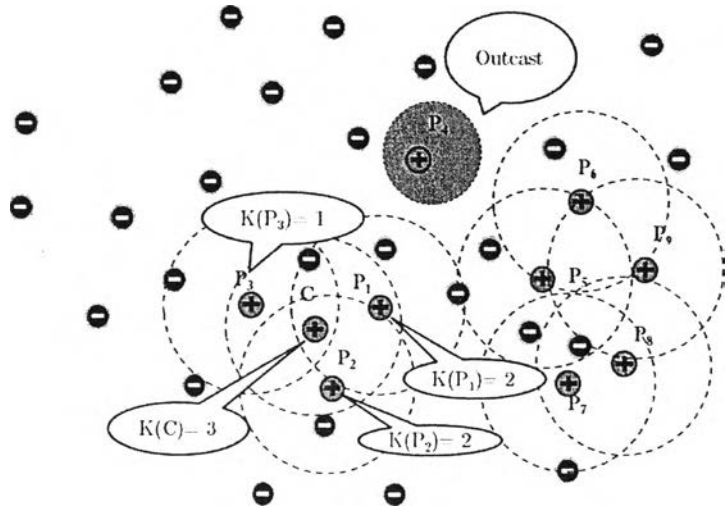

Figure 18: A visualization of assigning the number of K process

Algorithm 4: Adaptive neighbor SMOTE

Data: A numerical-attribute binary class dataset D containing a set of positive instances P and a set of negative instances N and the value of c.
Result: A nearly balanced dataset which is a combination of D and a set of synthetic instances S, a set of minority outcast OC.

1. Initialization $t = 1$;
2. $\{Pused, OC, SL, E\}$ = OutcastExtraction(D, P, C)
3. Define $\varepsilon = \max E$
4. For $p_i \in Pused$ do
5. Let $Np_i = \{p_j \in Pused | d(p_i, p_j) \leq \varepsilon\}$
6. Let $k_i = |Np_i|$
7. End for
8. While $t < \frac{|N|}{|Pused|}$ do
9. For $p_i \in Pused$ do
10. Randomly select $\hat{p}_i$ from Np_i
11. gap = GapCalculate(1, 1)
12. $p' = p_i + gap \times (\hat{p}_i - p_i)$
13. Add p' into S.
14. End for
15. $t = t + 1$
16. End while.

This choice of k_i selection is not the optimal choice to provide the best accuracy performance. However, it avoids multiple repeating runs for tuning the optimal value of k. Since the number of positive instances is relatively low comparing to the total number of instances, the distance and neighbor calculation take time and resource.

Moreover, the accuracy performance of adaptive neighbor synthetic oversampling process (ANS) is enhanced by making use of excluded minority outcasts. In the previous approach, minority outcasts taken out during the ANS process are brought back to perform minority outcast handling. As result, a new oversampling technique called **adaptive neighbors Synthetic Minority Oversampling TEchnique under 1NN outcast handling (ANSO)** is developed from the combination of minority outcast handling and ANS process. The flowchart of the

whole process of ANSO is shown in figure 19. The effectiveness of ANSO is presented through the experiment in chapter 4.

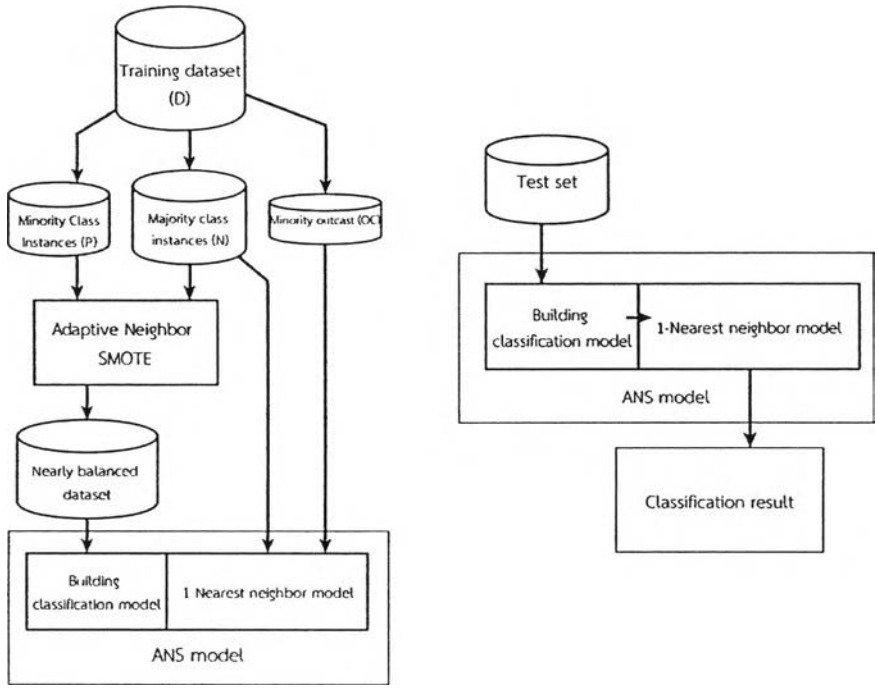

Figure 19: The flowchart of Adaptive neighbors Synthetic Minority Oversampling TEchnique under 1-NN outcast handling

3.5 The time complexity analysis

To calculate the time complexity of suggested oversampling techniques and some existing oversampling techniques i.e. SMOTE and safe-level SMOTE, this section uses n as the number of instances in a dataset and p is the number of positive instances. Each instance is complete with no missing value having d attributes. The parameter value of k and c are both set as 5. These values are set based on the setting given in the papers of SMOTE and safe-level SMOTE.

For SMOTE, the first stage is to find k nearest neighbors of p positive instances. This nearest neighbor process contains two major steps. The first one is to calculate the distance matrix of p positive instances. This step requires the time complexity of $O(p^2 \cdot d)$. The other step is to determine the least k distance values and the indices which provide these k values. This step requires the time complexity of $O(p^2 \cdot k)$. The combined time for the k-nearest neighbor process is $O(p^2 \cdot d + p^2 \cdot k)$. The next stage for SMOTE is to generate new synthetic instances. For each iteration, the

algorithm has to go through every p positive instance and synthesizes a new instance. It takes $O(p \cdot d)$ for processing one iteration. Since the oversampling is needed to be done until the dataset is nearly balanced, so the number of iterations is at least n/p. The time complexity for this synthetic generating process becomes $O((n/p) \cdot p \cdot d)$ or $O(n \cdot d)$. Adding to the time from previous stage, the total time complexity of SMOTE is $O(p^2 \cdot d + p^2 \cdot k + n \cdot d)$. The biggest term of this time complexity is between $O(p^2 \cdot d)$ and $O(n \cdot d)$. Since $p^2 \cdot d = \alpha^2 \cdot n^2 \cdot d > n \cdot d$ if $\alpha^2 \cdot n > 1$, where α is the ratio of p over n. Then, the time complexity of SMOTE is $O(p^2)$.

Safe-level SMOTE contains the c-nearest neighbor process in order to calculate the safe-level of p positive instances. To find c nearest neighbors from n instances of p positive instances, the process requires the distance matrix calculation which costs $O(n \cdot p \cdot d)$ and the distance selection which costs $O(n \cdot p \cdot k)$. So the total time complexity of this stage becomes $O(n \cdot p \cdot d + n \cdot p \cdot k)$. After this process, the entire SMOTE algorithm is performed. Then, the total time complexity of safe-level SMOTE is $O(n \cdot p \cdot d + n \cdot p \cdot k + p^2 \cdot d + p^2 \cdot k + n \cdot d)$ which equals to $O(n \cdot p)$. This time complexity of safe-level SMOTE is larger than one from SMOTE depending on the imbalance of the dataset.

For relocating safe-level SMOTE, the entire process of safe-level SMOTE is performed here except for the relocating process which is added from original safe-level SMOTE. The relocating process requires the extra distance calculation from the synthetic instance to negative neighbors of two positive instances which are used to generate. The maximum number of negative neighbors is $2 \cdot (k - 1)$, so the time complexity for this stage is $O(2 \cdot (k - 1) \cdot p \cdot d)$. Including the time complexity for safe-level SMOTE, the total time complexity for relocating safe-level SMOTE is $O(n \cdot p \cdot d + n \cdot p \cdot k + p^2 \cdot d + p^2 \cdot k + n \cdot d + 2 \cdot (k - 1) \cdot p \cdot d)$ which also equals to $O(n \cdot p)$ since either k (the default setting is 5) or d is much less than both p and n.

For adaptive neighbor SMOTE, the same c-nearest neighbor as one in safe-level SMOTE is performed to detect the minority outcast giving the time complexity as $O(n \cdot p \cdot d + n \cdot p \cdot k)$. The calculation for *Eps* which is used to determine the dynamic k for each positive value can be included into the distance sorting step of this process, so there is no additional time for this calculation. However, the counting leading to the number of k for each positive number requires the extra $O(p)$. The same distance matrix calculation step for k-nearest neighbor stage of ANS is also performed; however, the sorting step becomes $O(k_{max} \cdot p^2)$, where $k_{max} = \max \{k_i \mid i = 1, 2, \ldots, p\}$. k_{max} is usually larger than 5 and sometimes exceeds the number of attributes d. The

synthetic generating process is the same as one of SMOTE, so it requires the same time complexity which is $O(n \cdot d)$. In conclusion, the total time complexity for adaptive neighbor SMOTE becomes $O(n \cdot p \cdot d + n \cdot p \cdot k + p + p^2 \cdot d + k_{max} \cdot p^2 + n \cdot d)$ which also equals to $O(n \cdot p)$. The comparison of time complexity from each oversampling technique is shown in table 5.

Table 5: The summary of time complexities of SMOTE, safe-level SMOTE and suggested oversampling techniques in the framework

	Time Complexity
SMOTE	$O(p^2 \cdot d + p^2 \cdot k + n \cdot d) = O(p^2)$
SLS	$O(n \cdot p \cdot d + n \cdot p \cdot k + p^2 \cdot d + p^2 \cdot k + n \cdot d) = O(n \cdot p)$
RSLS	$O(n \cdot p \cdot d + n \cdot p \cdot k + p^2 \cdot d + p^2 \cdot k + n \cdot d + 2 \cdot (k - 1) \cdot p \cdot d) = O(n \cdot p)$
ANS	$O(n \cdot p \cdot d + n \cdot p \cdot k + p + p^2 \cdot d + k_{max} \cdot p^2 + n \cdot d) = O(n \cdot p)$

Since p can be written in term of αn where α is less than 0.5, so either $O(p^2)$ or $O(n \cdot p)$ both equals to $O(n^2)$. Then, it can be concluded that there is not much difference between the time complexities of these oversampling techniques.