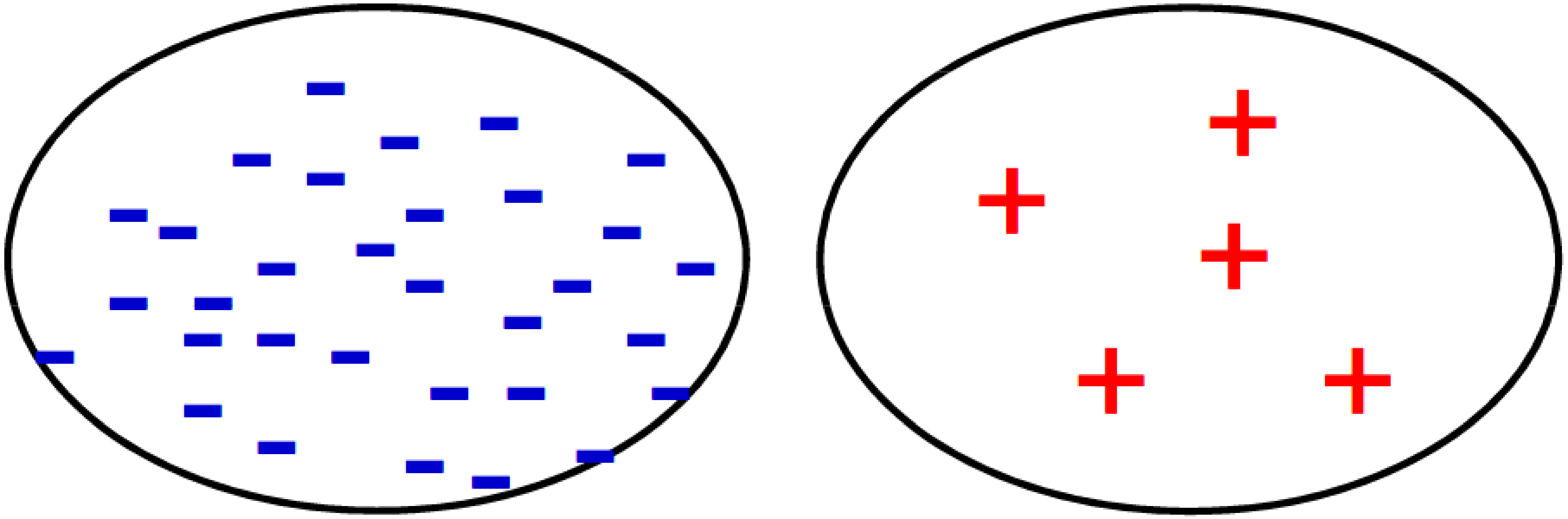
An introduction to unbalanced data classification

2018.08.02

张文涛

Concept

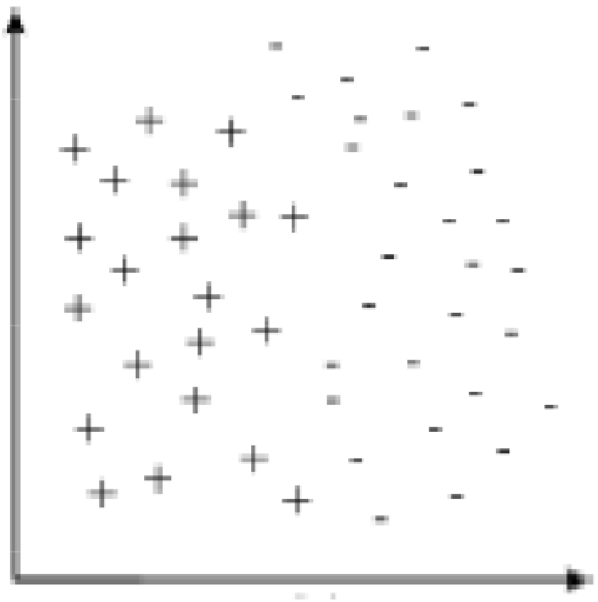
The data contains many more examples of one class than the other.



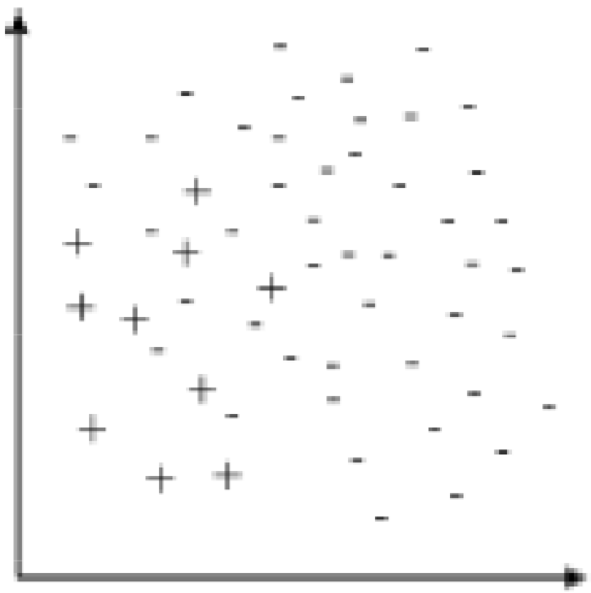
Background

- There exist many domains that do not have a balanced data set.
- There are a lot of problems where the most important knowledge usually resides in the minority class.
    - CTR estimation,
    - Anti fraud identification
- Most ML algorithms are designed to optimize overall accuracy without taking into account the relative distribution of each class.

Problem



An easier problem



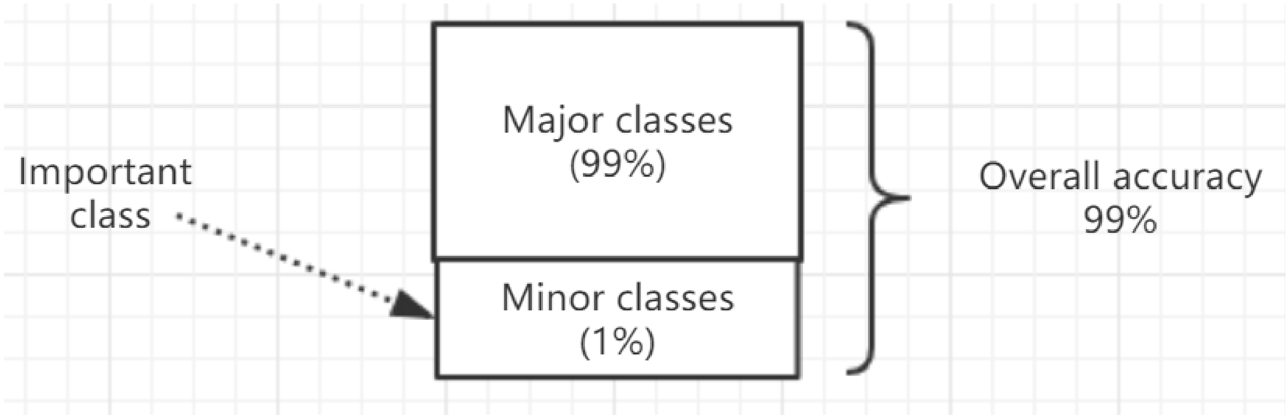
More difficult one

Majority classes overlaps the minority class:

- Ambiguous boundary between classes
- Influence o of n noisy examples
- The optimization goal of standard learners is generally the accuracy rate

Problem

- Standard learners are often biased towards the majority class
- Examples from the minority class tend to be misclassified
- The classifiers tend to ignore small classes while concentrating on classifying the large ones accurately.



If we predict all the data as a positive class, the overall accuracy is as high as 99%, but the auc is 50%

Evaluation

		True class	
		p	**n**
Hypothesized class	**Y**	**True Positives**	**False Positives**
	N	**False Negatives**	**True Negatives**
Column totals:		**P**	**N**

$$\text{fp rate} = \frac{FP}{N} \qquad \text{tp rate} = \frac{TP}{P}$$

$$\text{precision} = \frac{TP}{TP+FP} \qquad \text{recall} = \frac{TP}{P}$$

$$\text{accuracy} = \frac{TP+TN}{P+N}$$

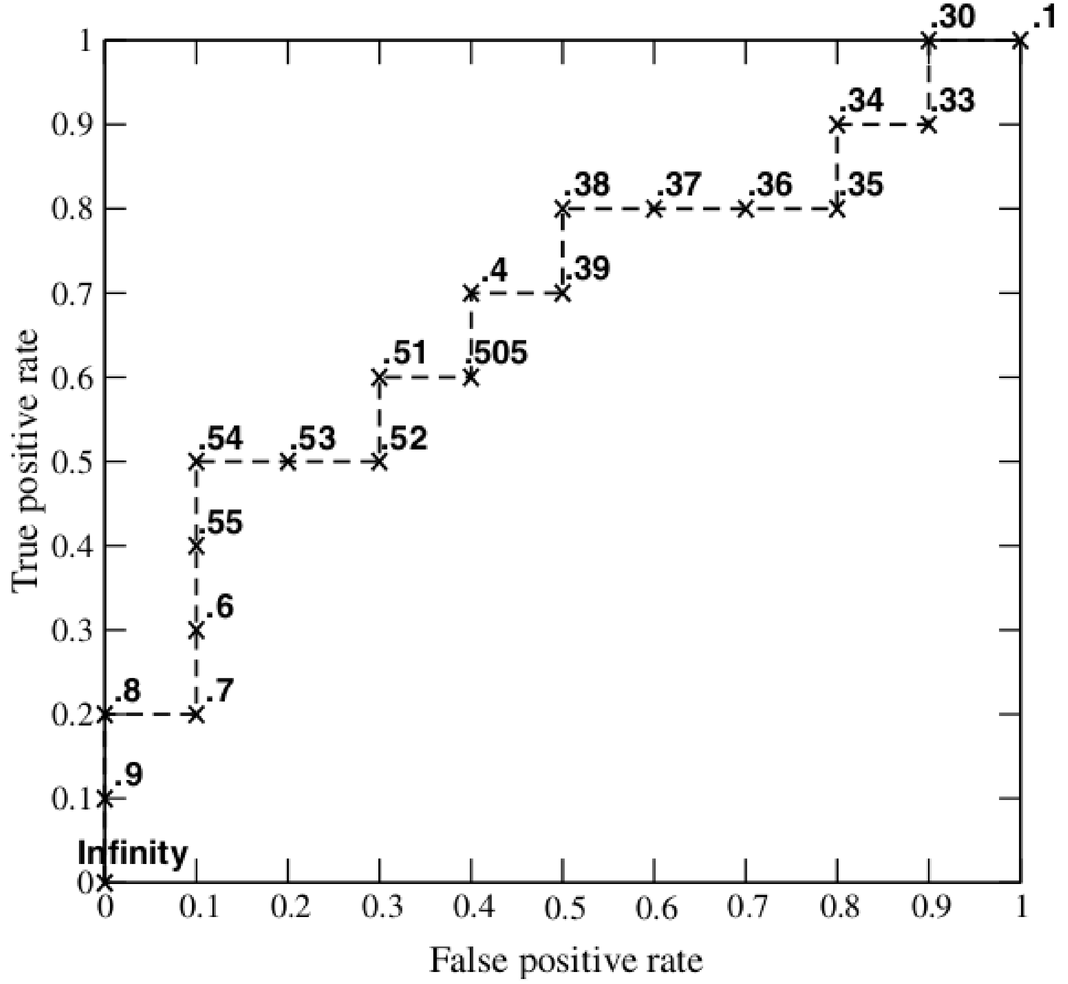
$$\text{F-measure} = \frac{2}{1/\text{precision}+1/\text{recall}}$$

Fig. 1. Confusion matrix and common performance metrics calculated from it.

Evaluation

Inst#	Class	Score	Inst#	Class	Score
1	**p**	.9	11	**p**	.4
2	**p**	.8	12	**n**	.39
3	**n**	.7	13	**p**	.38
4	**p**	.6	14	**n**	.37
5	**p**	.55	15	**n**	.36
6	**p**	.54	16	**n**	.35
7	**n**	.53	17	**p**	.34
8	**n**	.52	18	**n**	.33
9	**p**	.51	19	**p**	.30
10	**n**	.505	20	**n**	.1

The probability of each sample being predicted as a positive sample

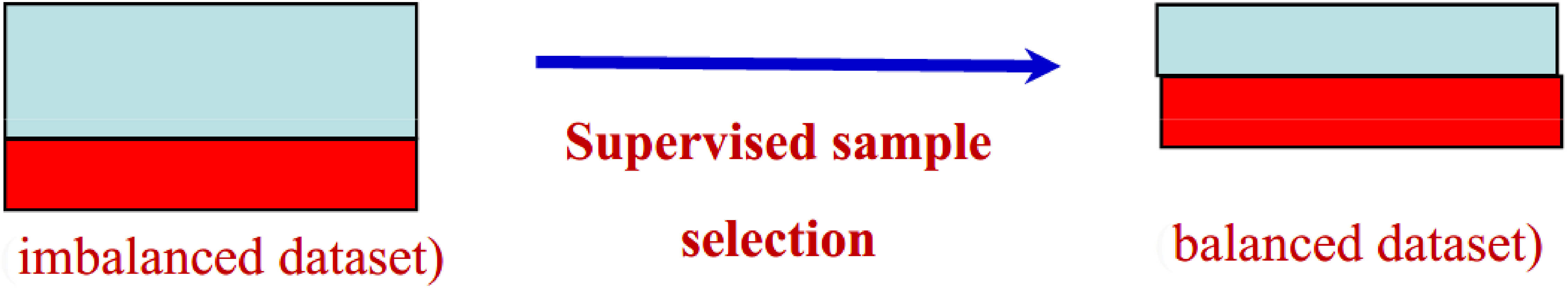


The ROC curve

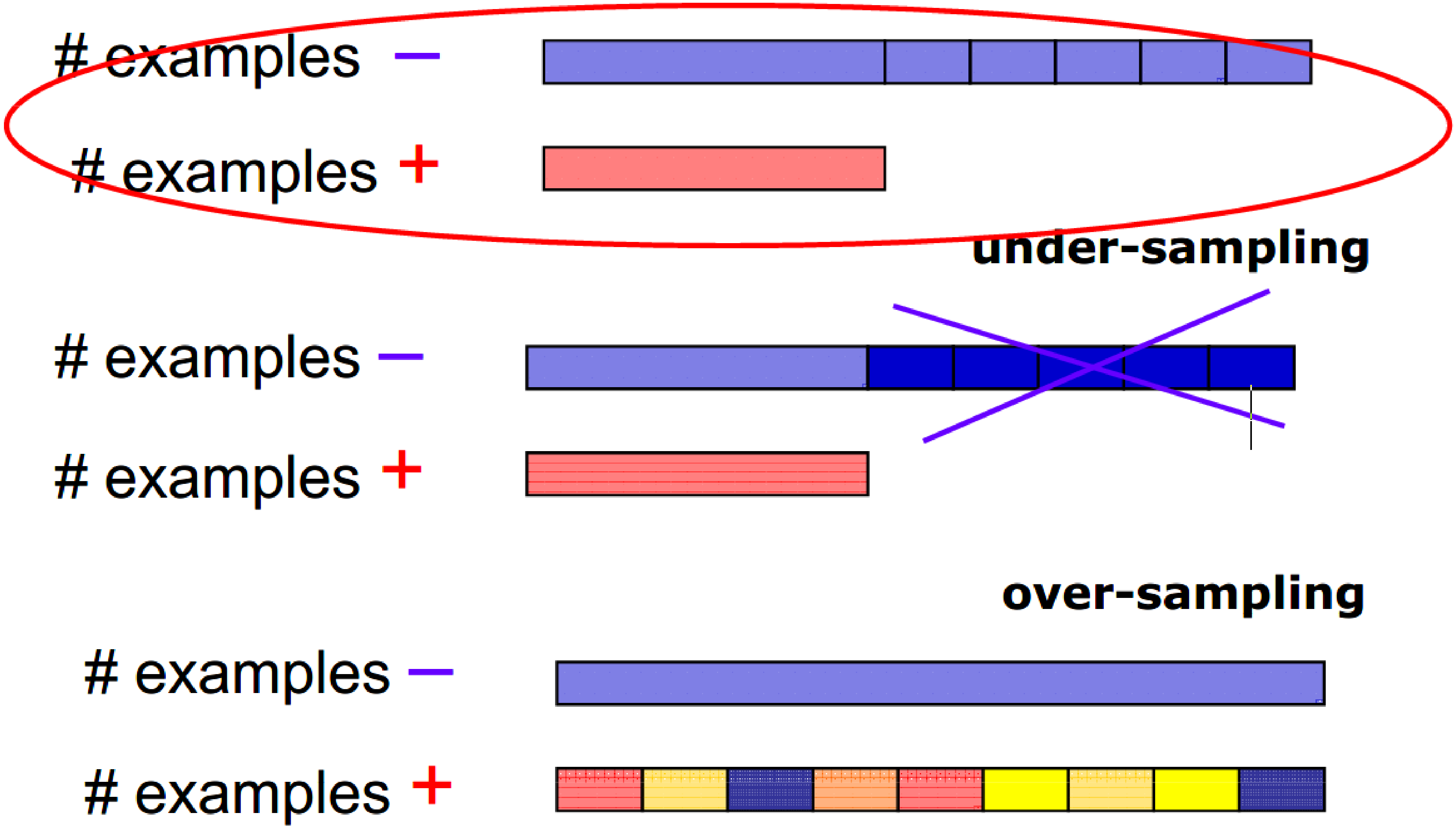
Strategies to deal with imbalanced data sets

Resample

Resampling is the process of manipulating the distribution of the training examples in an effort to improve the performance of classifiers.



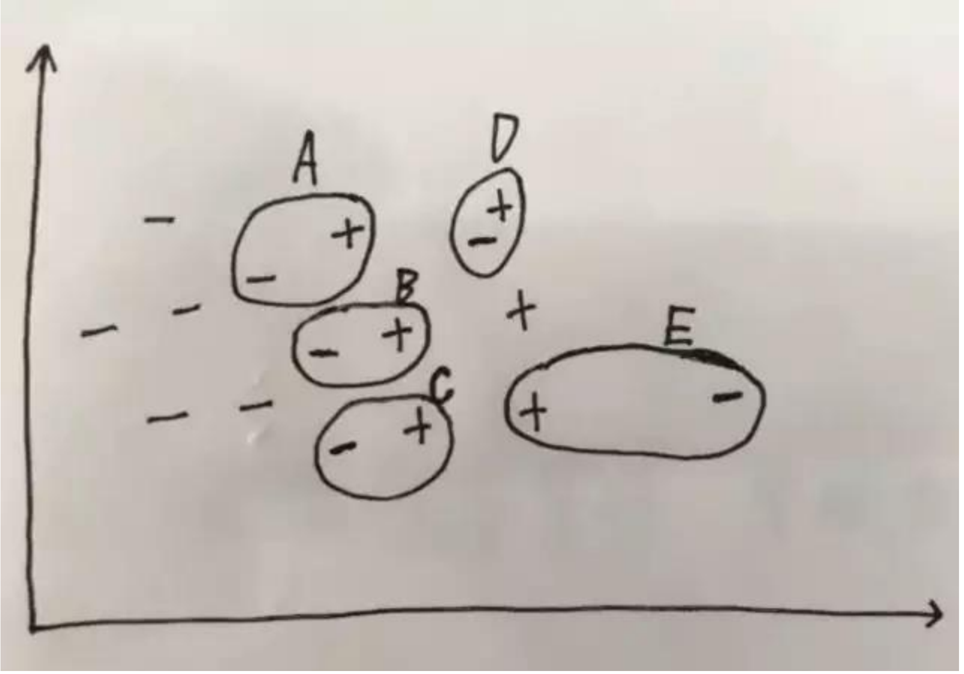
Undersampling vs oversampling



Under-sampling

Tomek Links

- Idea:
  - To remove both noise and borderline examples of the majority class
- Definition:
  - E_i, E_j belong to different classes
  - $d(E_i, E_j)$ is the he distance between them
  - $A(E_i, E_j)$ pair is called a Tomek link if there is no example EI, such that $d(E_i, EI) < d(E_i, E_j)$ or $d(E_j, EI) < d(E_i, E_j)$

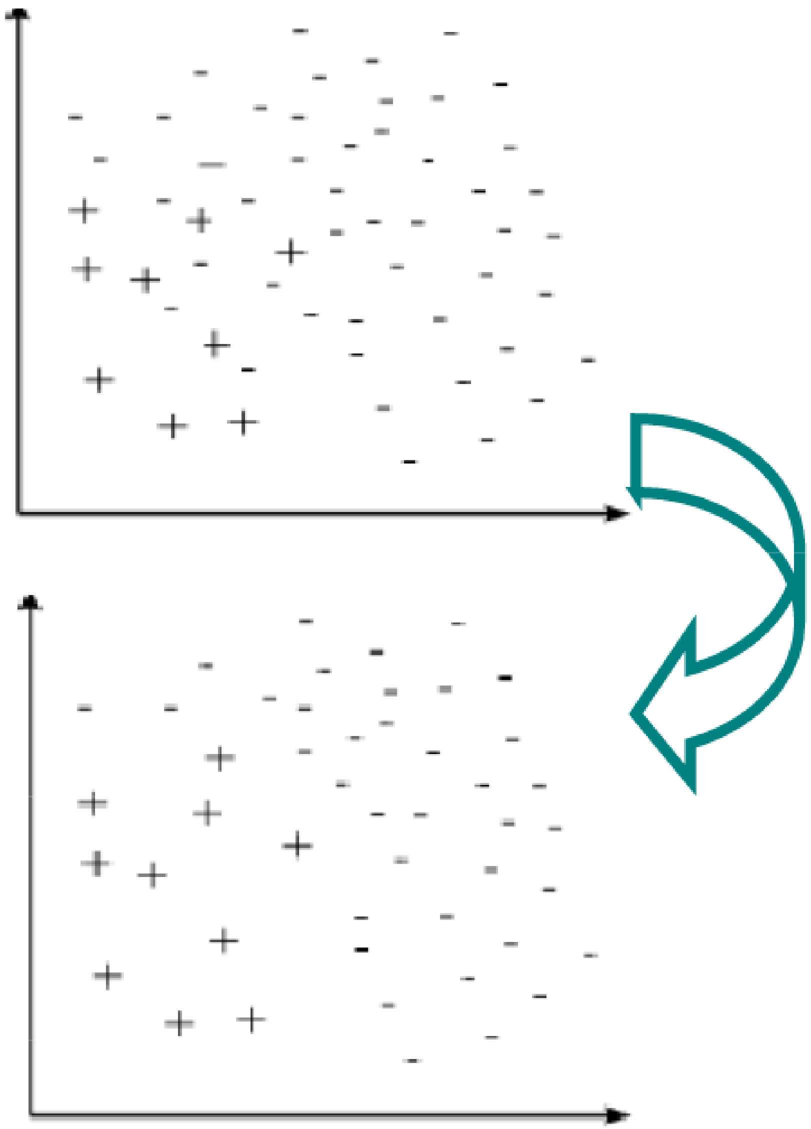


Under-sampling

Usage of Tomek Links

- Under sampling
  - Remove the samples of most classes that belong to Tomek Links

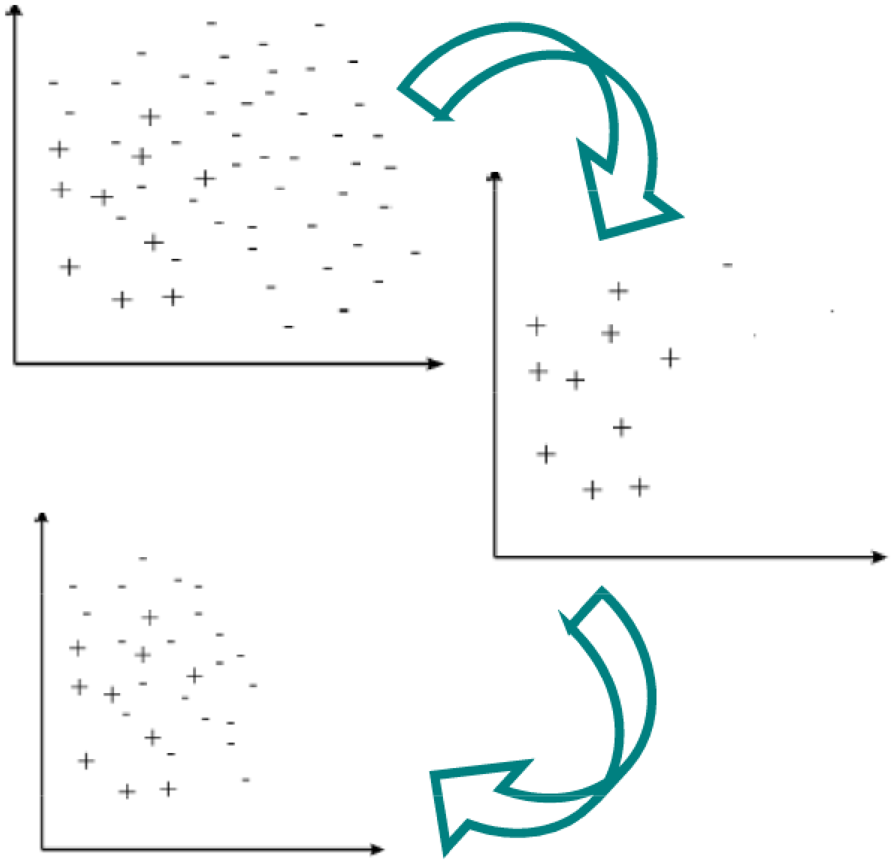
- Data cleaning
  - Remove the samples that belong to Tomek Links



Under-sampling

1NN

- Idea
  - remove both noise and borderline examples
- Definition:
  - Let A be the original training set
  - Let B contains all positive examples from A and one randomly selected negative example
  - Classify A with the 1-NN rule using
  - the examples in B
  - Move all misclassified example from A to B



Under-sampling

NearMiss

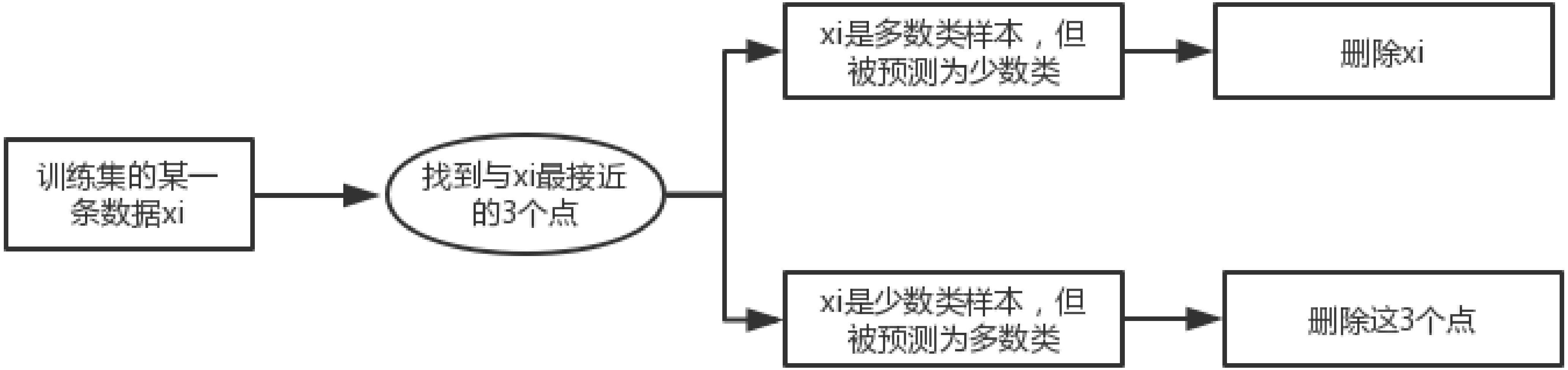
- **Idea**
  - Remove majority class samples by distance
- **NearMiss-1**
- Remove the majority sample which has the smallest average distance from the nearest 3 minority class samples
- **NearMiss-2**
- Remove the majority sample which has the smallest average distance from the largest 3 minority class samples
- **NearMiss-3**
- For each minority sample, select a fixed number of nearest majority class samples

Under-sampling

NCL

➢ Idea

- Emphasize more data cleaning than data reduction



Oversampling

Random oversampling

- Idea
    - Randomly replicating examples
- **Advantage**
- simple to implement
- **Disadvantage**
- Too many repeated samples may lead to overfitting

Oversampling

Smote

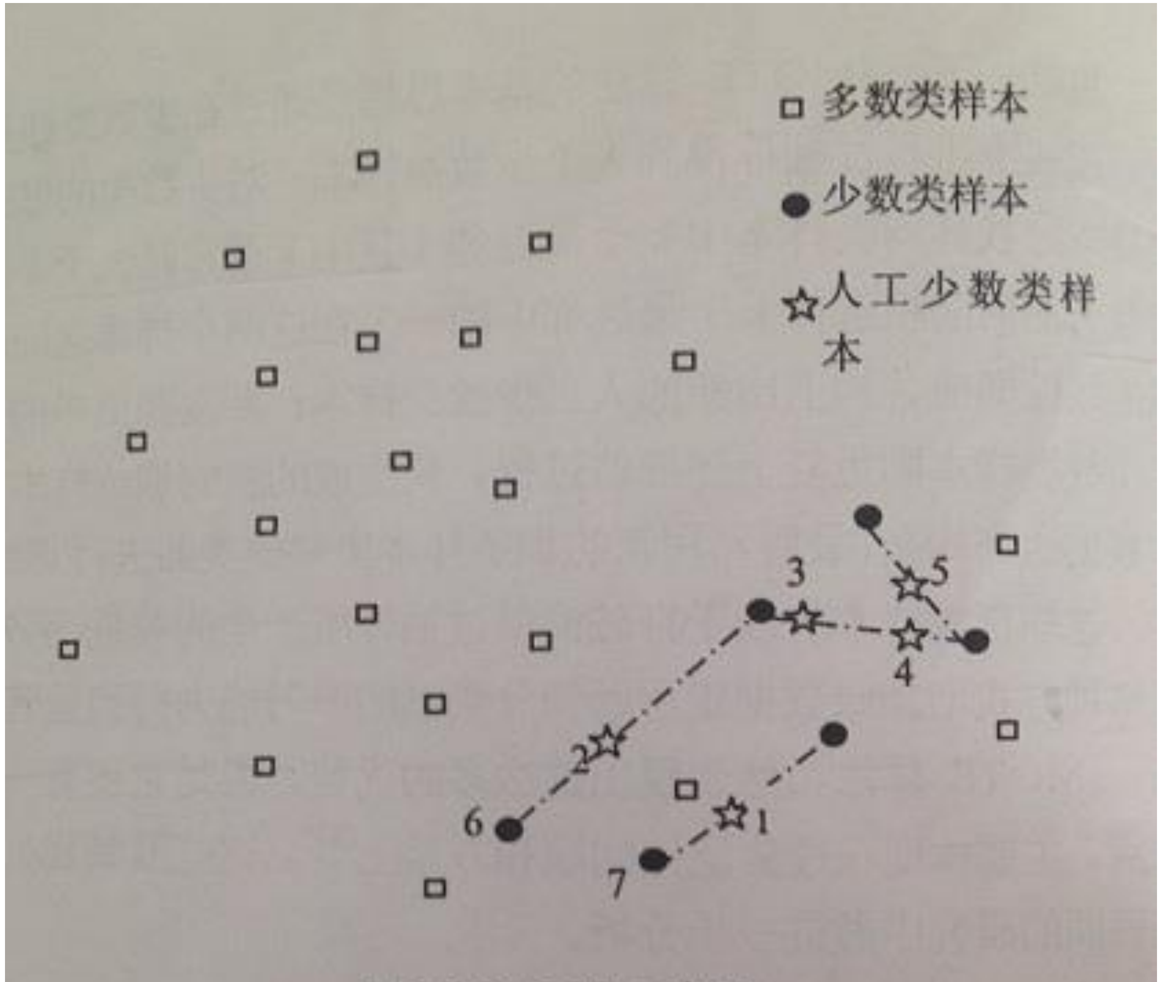
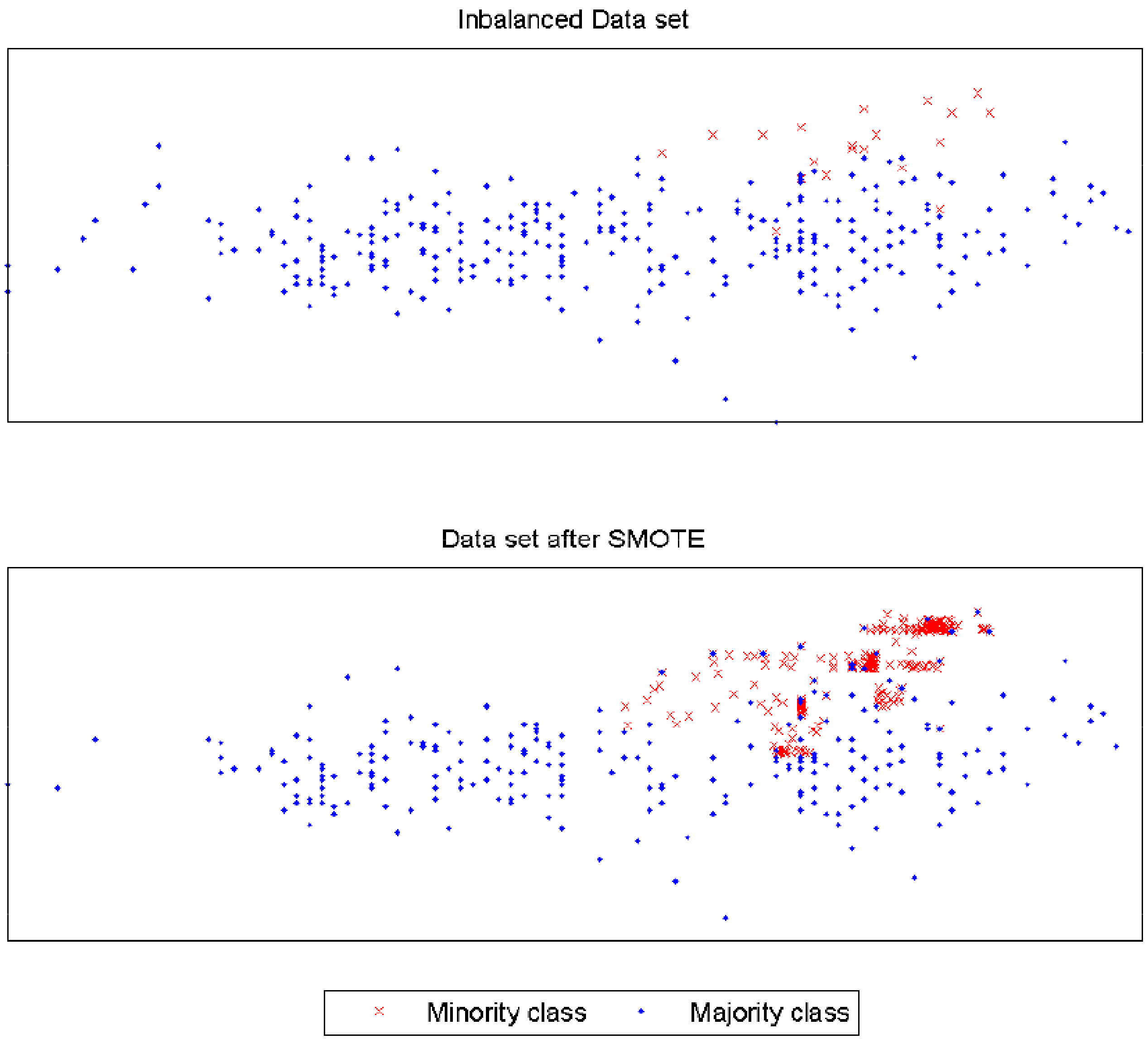


图 3.3 SMOTE 示意图

Consider a sample (6,4) and let (4,3) be its nearest neighbor.
(6,4) is the sample for which k-nearest neighbors are being identified
(4,3) is one of its k-nearest neighbors.
Let:
f1_1 = 6 f2_1 = 4 f2_1 - f1_1 = -2
f1_2 = 4 f2_2 = 3 f2_2 - f1_2 = -1
The new samples will be generated as
(f1',f2') = (6,4) + rand(0-1) * (-2,-1)
rand(0-1) generates a random number between 0 and 1.

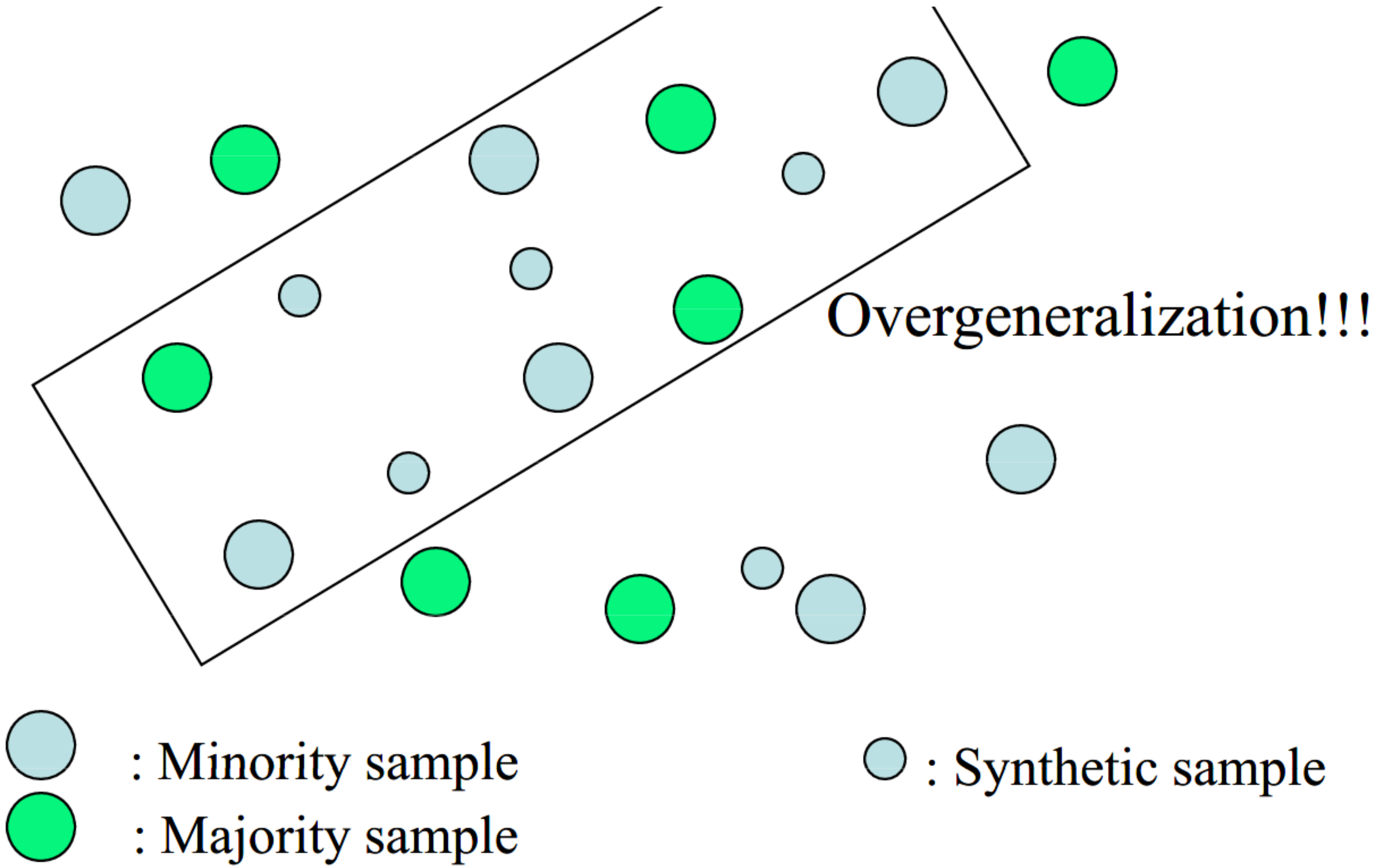
Oversampling

Smote

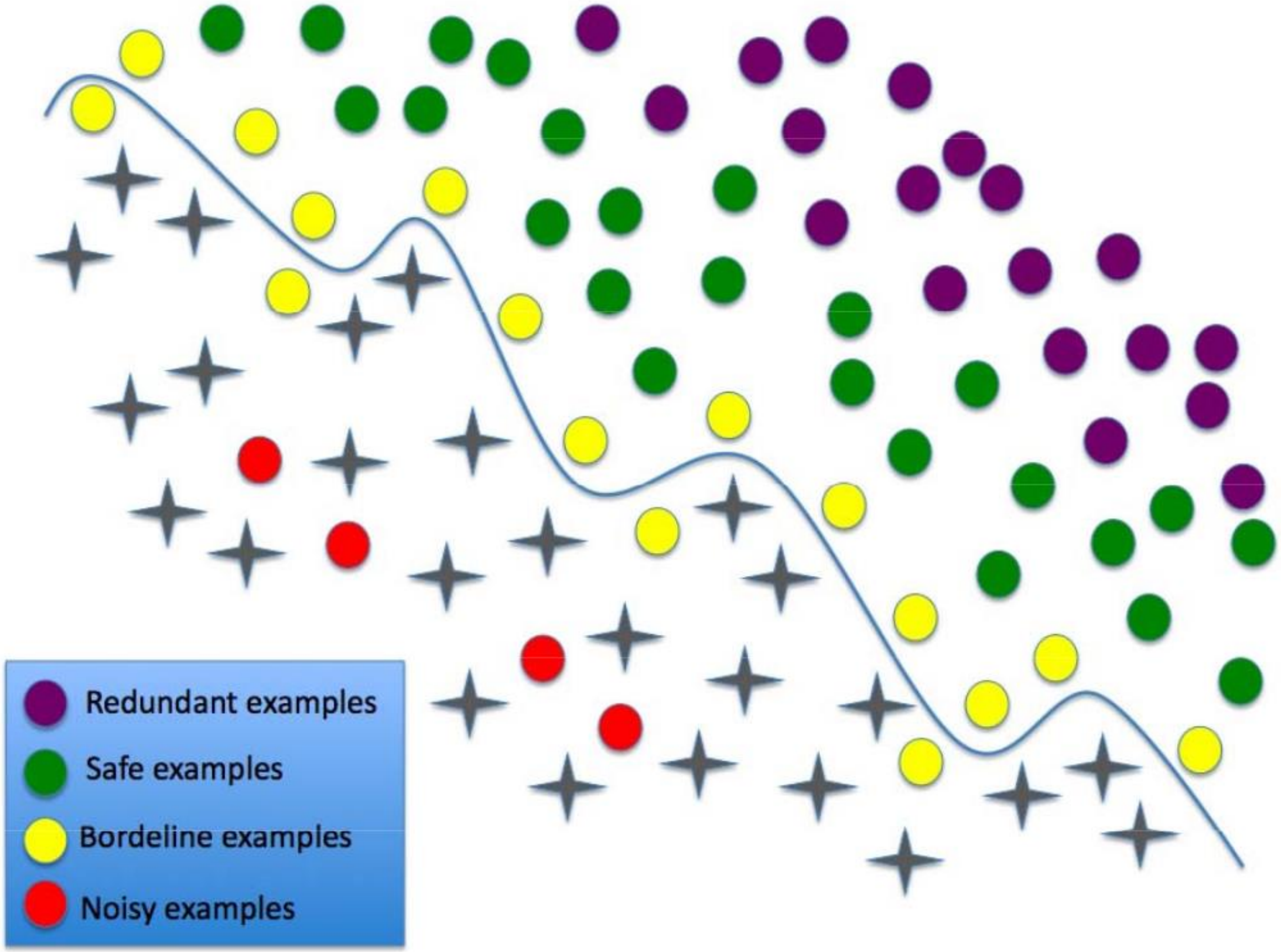


Oversampling

Smote Shortcomings

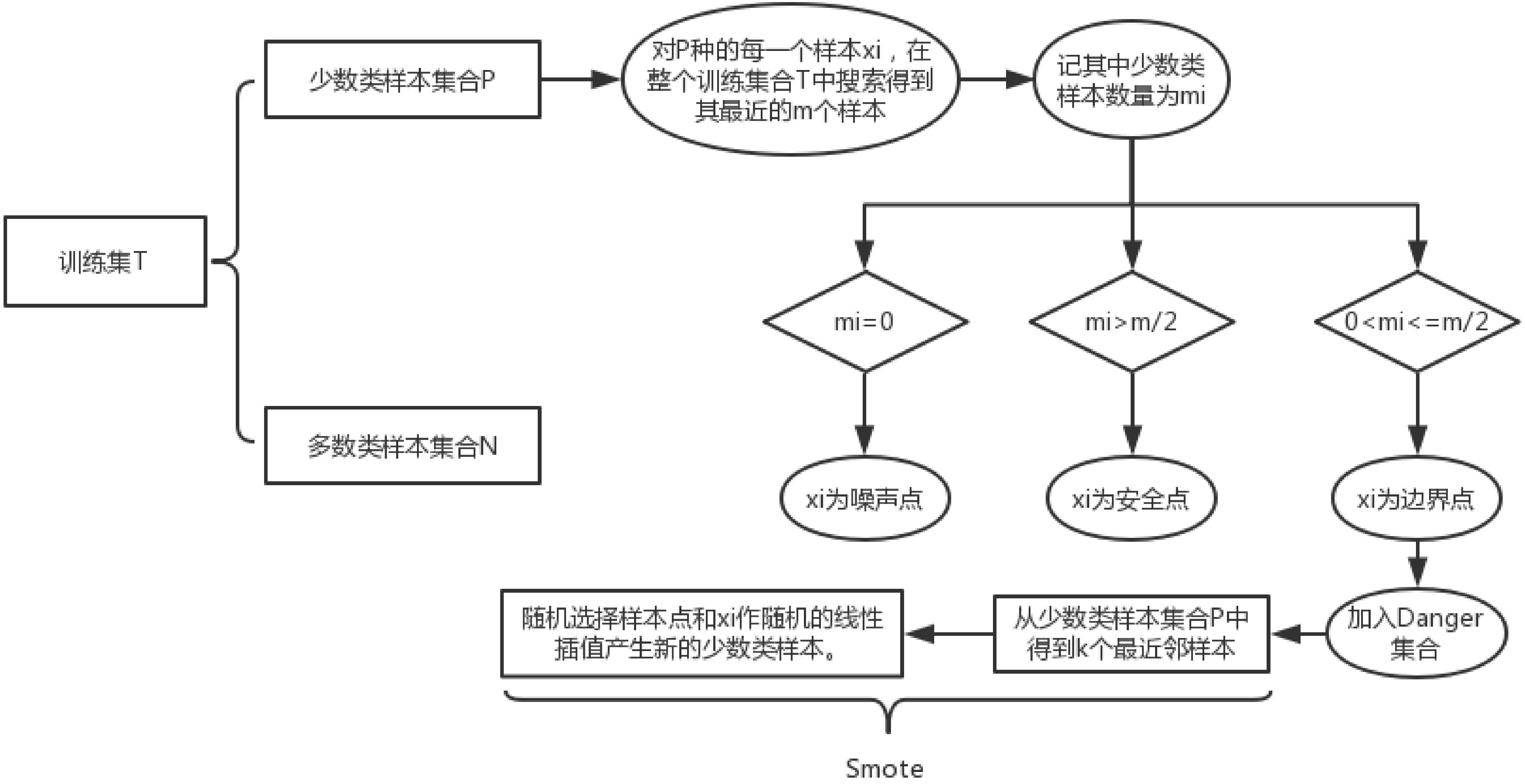


Oversampling



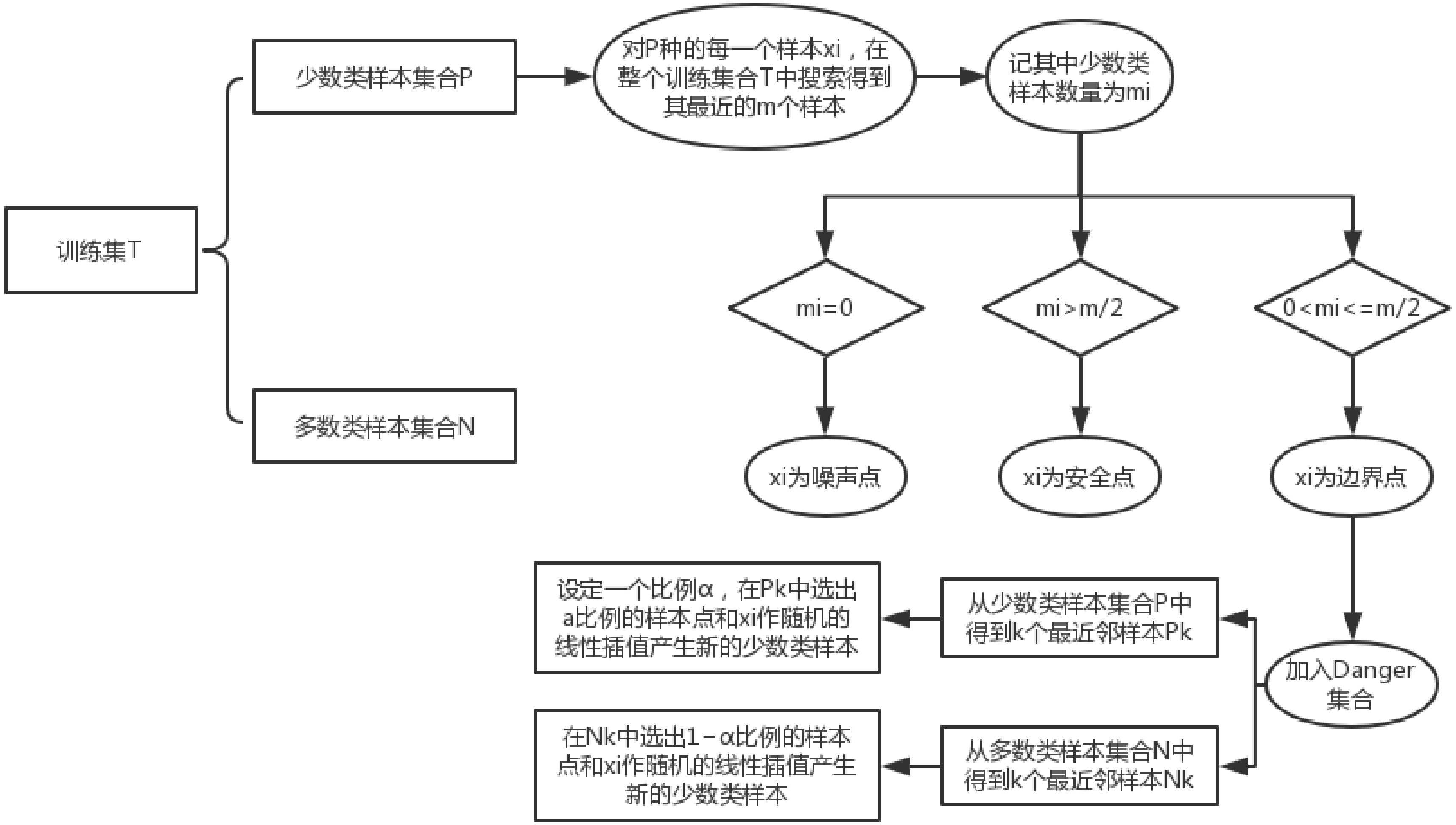
Oversampling

Borderline SMOTE-1



Oversampling

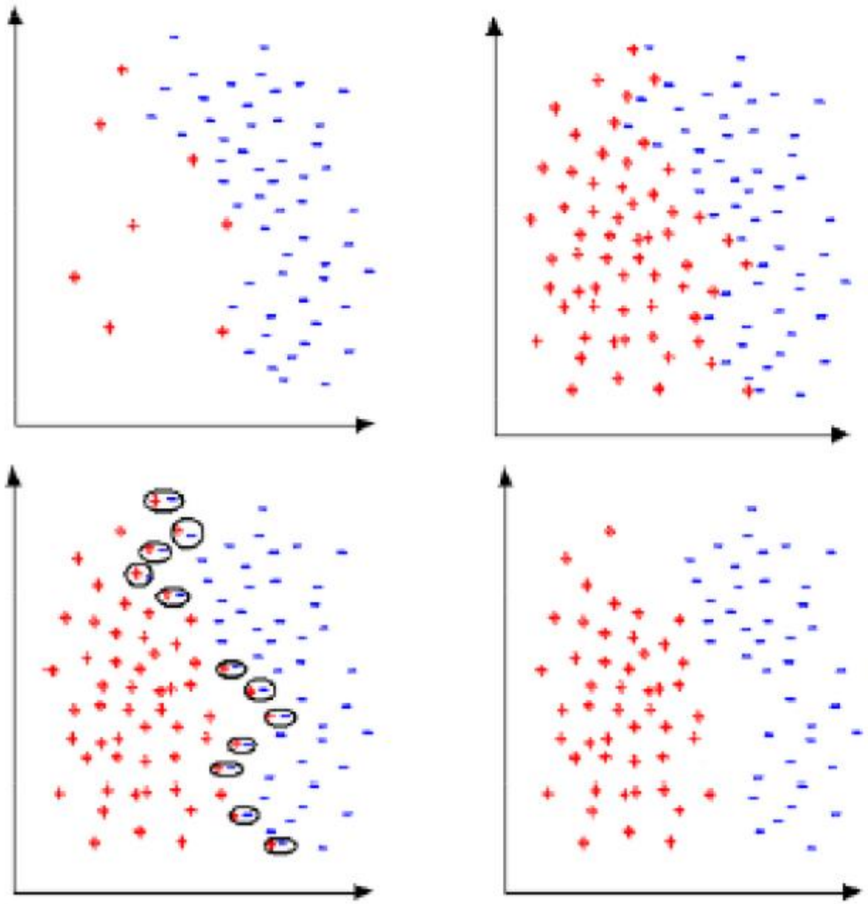
Borderline SMOTE-2



Oversampling

SMOTE+Tomek links

Figure: SMOTE+TomekLink



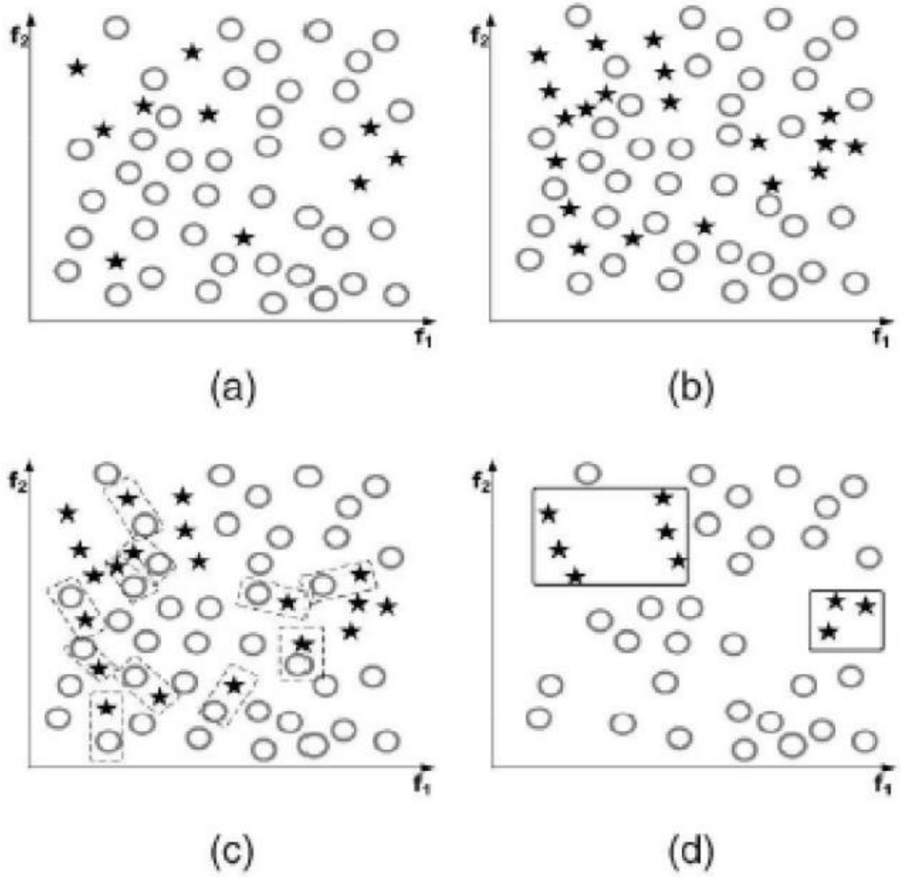
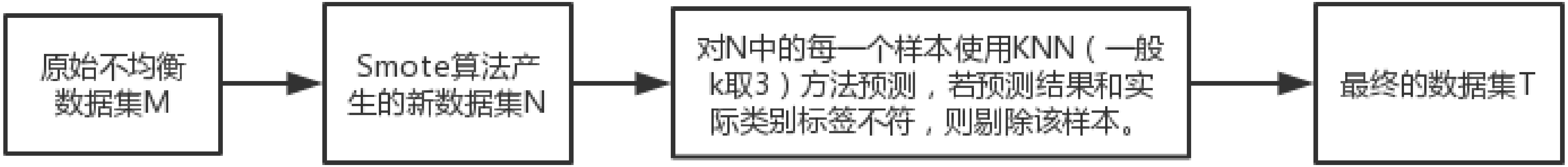


Figure 17: (a) Original data-set distribution. (b) Post-SMOTE data-set. (c) The identified Tomek Links. (d) The data-set after removing Tomek links

Oversampling

SMOTE+KNN



Cost-sensitive learning

- **Weighting the data space (data level):**
  - Change the distribution of the training sets (translation theorem)
  - Modifying final decision thresholds

- **Making a specific classifier learning algorithm cost-sensitive (algorithm level)**
  - Change the inner way the classifier works
  - Use a boosting approach

Ensemble

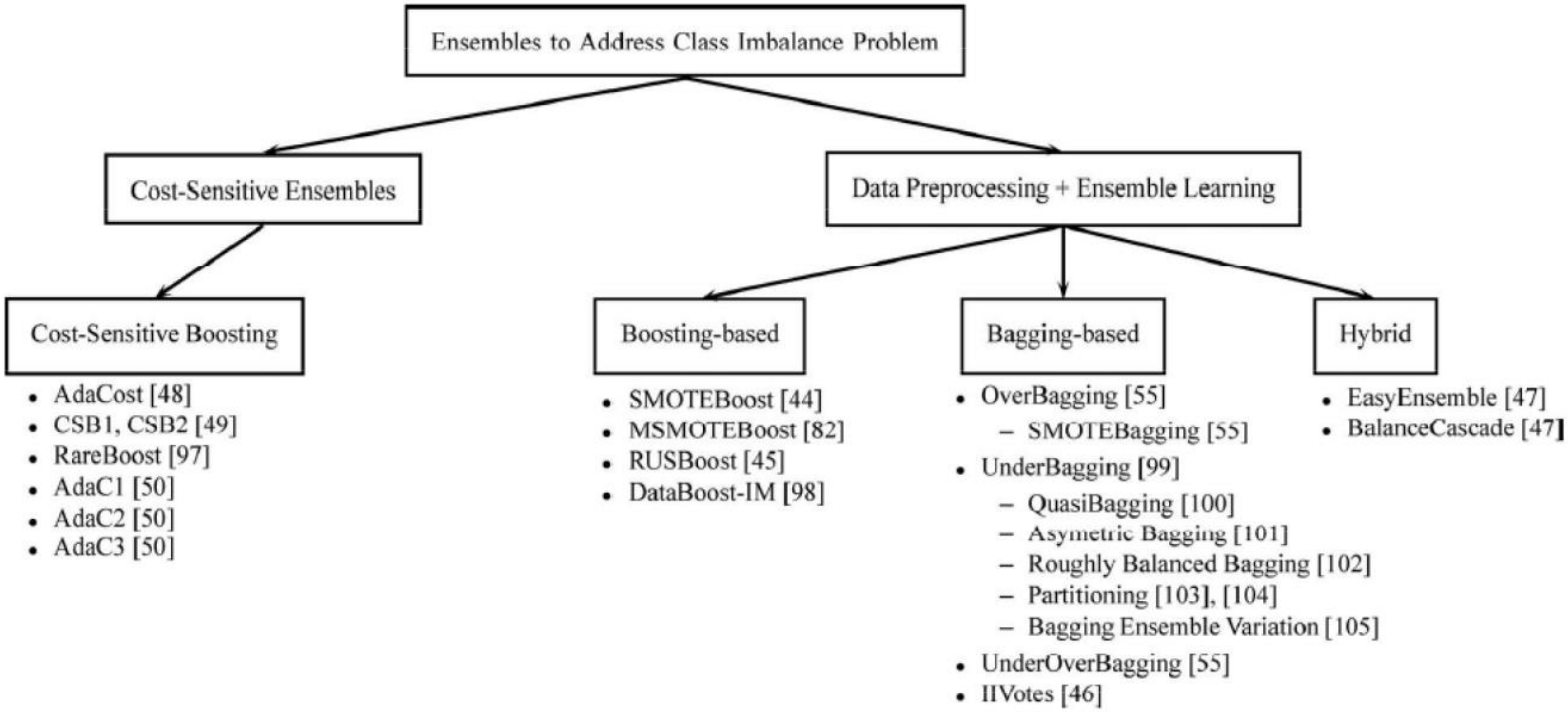


Fig. 3. Proposed taxonomy for ensembles to address the class imbalance problem.

Ensemble

- **Easy ensemble**

Algorithm 1 The `EasyEnsemble` algorithm.

1: {Input: A set of minority class examples $\mathcal{P}$, a set of majority class examples $\mathcal{N}$, $|\mathcal{P}| < |\mathcal{N}|$, the number of subsets T to sample from $\mathcal{N}$, and s_i, the number of iterations to train an AdaBoost ensemble H_i}

2: $i \Leftarrow 0$

3: **repeat**

4: $i \Leftarrow i + 1$

5: Randomly sample a subset $\mathcal{N}_i$ from $\mathcal{N}$, $|\mathcal{N}_i| = |\mathcal{P}|$.

6: Learn H_i using $\mathcal{P}$ and $\mathcal{N}_i$. H_i is an AdaBoost ensemble with s_i weak classifiers $h_{i,j}$ and corresponding weights $\alpha_{i,j}$. The ensemble's threshold is θ_i, i.e.,

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

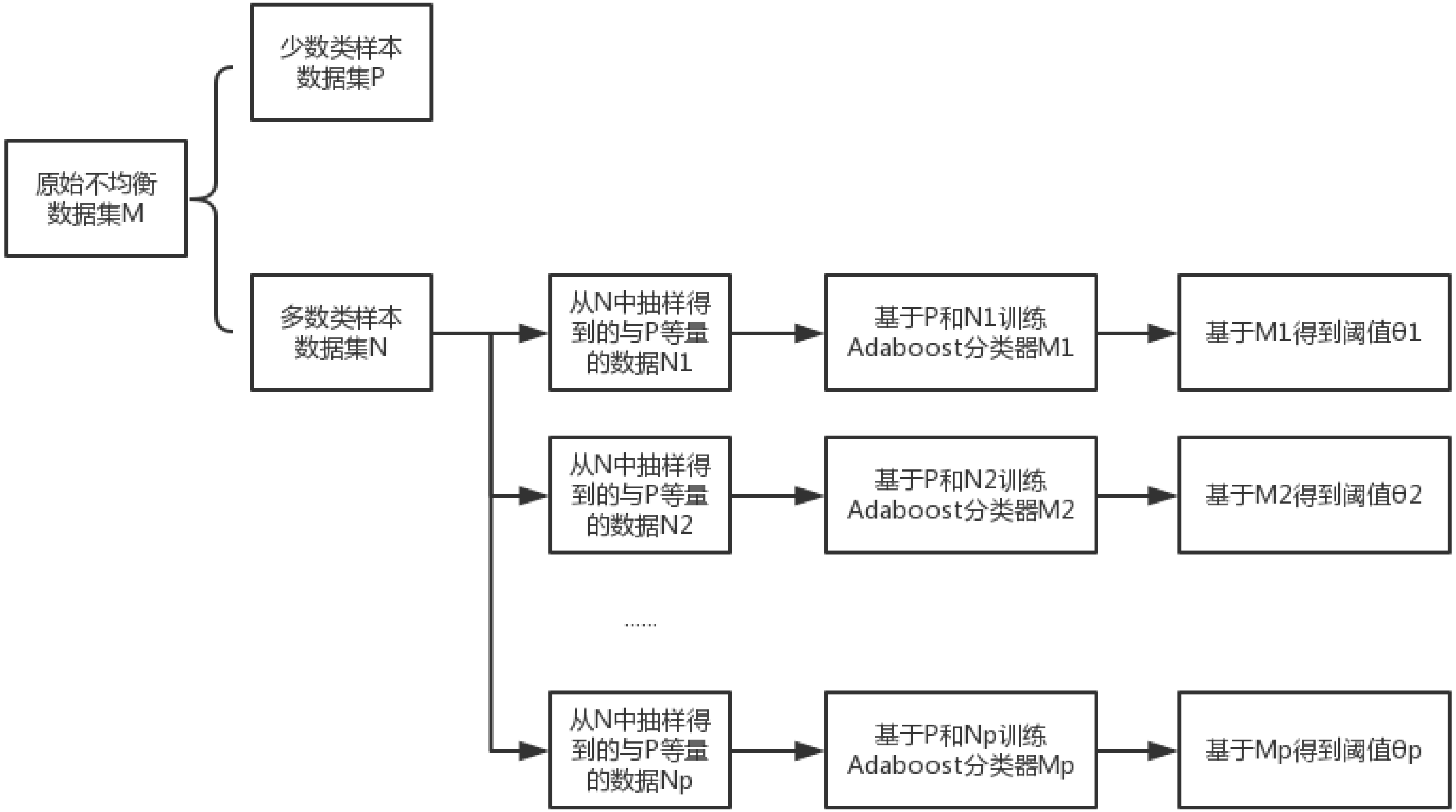
7: **until** $i = T$

8: Output: An ensemble

$$H(x) = \operatorname{sgn}\left(\sum_{i=1}^{T}\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \sum_{i=1}^{T} \theta_i\right).$$

Ensemble

➢ **Easy ensemble**



Ensemble

- **balance cascade**

Algorithm 2 The `BalanceCascade` algorithm.

1: {Input: A set of minority class examples $\mathcal{P}$, a set of majority class examples $\mathcal{N}$, $|\mathcal{P}| < |\mathcal{N}|$, the number of subsets T to sample from $\mathcal{N}$, and s_i, the number of iterations to train an AdaBoost ensemble H_i}

2: $i \Leftarrow 0$, $f \Leftarrow \sqrt[T-1]{|\mathcal{P}|/|\mathcal{N}|}$, f is the false positive rate (the error rate of misclassifying a majority class example to the minority class) that H_i should achieve.

3: **repeat**

4: $i \Leftarrow i + 1$

5: Randomly sample a subset $\mathcal{N}_i$ from $\mathcal{N}$, $|\mathcal{N}_i| = |\mathcal{P}|$.

6: Learn H_i using $\mathcal{P}$ and $\mathcal{N}_i$. H_i is an AdaBoost ensemble with s_i weak classifiers $h_{i,j}$ and corresponding weights $\alpha_{i,j}$. The ensemble's threshold is θ_i i.e.,

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

7: Adjust θ_i such that H_i's false positive rate is f.

8: Remove from $\mathcal{N}$ all examples that are correctly classified by H_i.

9: **until** $i = T$

10: Output: A single ensemble

$$H(x) = \operatorname{sgn}\left(\sum_{i=1}^{T}\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \sum_{i=1}^{T} \theta_i\right).$$

After T-1 epochs, the number of majority class samples is

$$|N| * f^{T-1} = |P|$$

Ensemble

➢ **CUSboost**

Algorithm 1 CUSBoost Algorithm

Input: Imbalanced data, D, number of iterations, k, and C4.5 decision tree induction algorithm.
Output: An ensemble model.
Method:

1: initialize weight, $x_i \in D$ to $\frac{1}{d}$;
2: **for** i = 1 to k **do**
3: create balanced dataset D_i with distribution D using cluster-based under-sampling;
4: derive a tree, M_i from D_i employing C4.5 algorithm;
5: compute the error rate of M_i, $error(M_i)$;
6: **if** $error(M_i) \geq 0.5$ **then**
7: go back to step 3 and try again;
8: **end if**
9: **for** each $x_i \in D_i$ that correctly classified **do**
10: multiply weight of x_i by $(\frac{error(M_i)}{1-error(M_i)})$; // update weights
11: **end for**
12: normalise the weight of each instances, x_i;
13: **end for**

To use the ensemble to classify instance, x_{New}:

1: initialise weight of each class to 0;
2: **for** i = 1 to k **do**
3: $w_i = log\frac{1-error(M_i)}{error(M_i)}$; // weight of the classifier's vote
4: $c = M_i(x_{New})$; // class prediction by M_i
5: add w_i to weight for class c;
6: **end for**
7: return the class with largest weight;

Ensemble

➢ CUSboost