Boosting for Learning Multiple Classes with Imbalanced Class Distribution

Yanmin Sun, Mohamed S. Kamel and Yang Wang

The Sixth International Conference on Data Mining (ICDM'06)

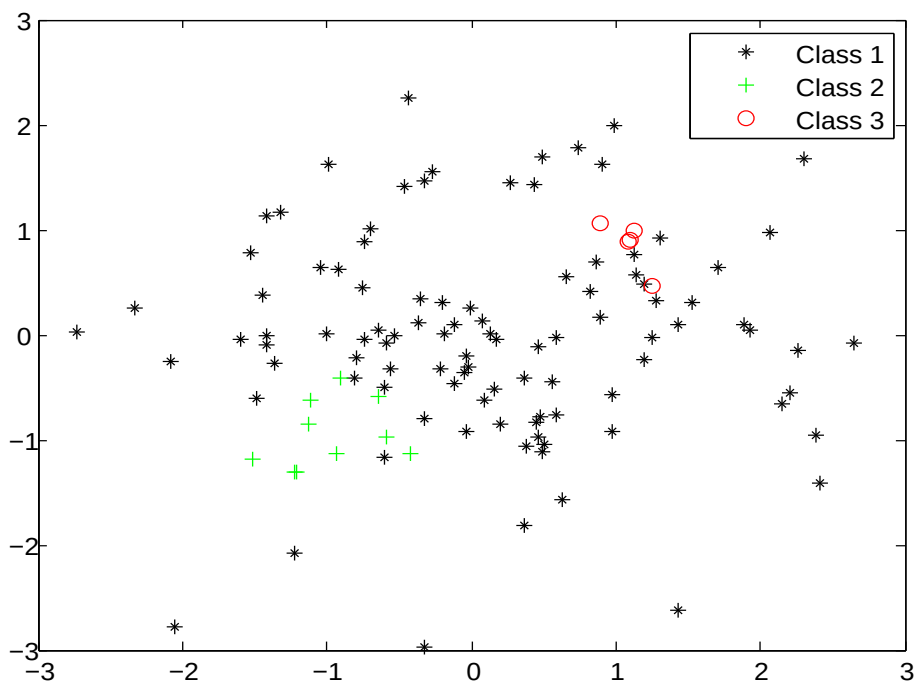
Presenter: Minhua Chen

Agenda

- The Class Imbalance Problem
- The Basic AdaBoost Algorithm
- Extending AdaBoost to Multi-class (AdaBoost.M1)
- Incorporating Cost-Sensitive Learning (AdaC2.M1)
- Experimental Results
- Conclusion

The Class Imbalance Problem: Introduction

- Some classes are represented by a large number of instances(samples) than other classes.
- Landmine detection, medical diagnosis, abnormality detection, etc.
- Direct classification may be biased towards the majority classes and result in poor performance on the minority classes.



The Class Imbalance Problem: Method

How to deal with the class imbalance problem?

- Resampling: *Oversample* data from the minority classes or *undersample* data from the majority classes.
- Cost-sensitive learning: Assign misclassification costs to data from each class, forcing the classifier to concentrate on the minority classes.
- AdaBoost: Change the underlying data distribution and classify in the re-weighted data space iteratively.

  Why AdaBoost is proper for the class imbalanced problem?
  – A platform to integrate the above two approaches.
  – Weak classifiers can be *boosted* to a stronger classifier.

The Basic AdaBoost Algorithm: Introduction

- AdaBoost (Freund and Schapire, 1995), short for Adaptive Boosting, calls a weak classifier iteratively, and finally outputs a stronger classifier by combining all the previously used weak classifiers.
- AdaBoost maintains a set of weights over the training set ($\boldsymbol{D}^t$), and updates the weights adaptively according to classification errors in each iteration.
- Misclassified samples tend to receive higher weights.
- The weak classifier is designed to minimize the weighted error

$$\epsilon_t = \mathsf{Pr}_{i \sim \boldsymbol{D}^t}[h_t(x_i) \neq y_i] = \sum_{i: h_t(x_i) \neq y_i} D^t(i)$$

The Basic AdaBoost Algorithm: AdaBoost

1. Given $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_m, y_m)$ where $\boldsymbol{x}_i \in \mathcal{X}$ are samples and $y_i \in \mathcal{Y} = \{-1, 1\}$ are corresponding labels.
2. Initialize sample weights $D^1(i) = \frac{1}{m} \quad (i = 1, 2, \cdots, m)$.
3. For $t = 1, 2, \cdots, T$
   - Train the weak classifier h_t using sample weights $\boldsymbol{D}^t$. $h_t(\mathcal{X}) \rightarrow \mathcal{Y} = \{-1, 1\}$.
   - Choose the weight update parameter $\alpha_t = \frac{1}{2} \log \left(\frac{\sum_{i:h_t(\boldsymbol{x}_i)=y_i} D^t(i)}{\sum_{i:h_t(\boldsymbol{x}_i)\neq y_i} D^t(i)} \right) = \frac{1}{2} \log \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$.
   - Update and normalize the sample weights:

$$D^{t+1}(i) = \frac{D^t(i) \exp\left(-\alpha_t I[h_t(\boldsymbol{x}_i) = y_i]\right)}{Z_t}$$

4. Output the final classifer $H(\boldsymbol{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}) \right)$.

The Basic AdaBoost Algorithm: Explanations

- $I[\cdot]$ is an indicator function: $I[h_t(\boldsymbol{x}_i) = y_i] = \begin{cases} +1 & \text{if } h_t(\boldsymbol{x}_i) = y_i \\ -1 & \text{if } h_t(\boldsymbol{x}_i) \neq y_i \end{cases}$

  and

$$Z_t = \sum_{i=1}^{m} D^t(i) \exp\left(-\alpha_t I[h_t(\boldsymbol{x}_i) = y_i]\right)$$

  is a normalization factor.

- The overall training error is upper bounded:

$$\frac{1}{m} \sum_{i=1}^{m} [H(\boldsymbol{x}_i) \neq y_i] \leq \prod_{t=1}^{T} Z_t$$

  where $[\cdot]$ is another indicator function with

$$[\text{Expression}] = \begin{cases} 1 & \text{if Expression is true} \\ 0 & \text{if Expression is false} \end{cases}$$

The Basic AdaBoost Algorithm: More Explanations

- How α_t is chosen? – by minimizing the error upper bound.

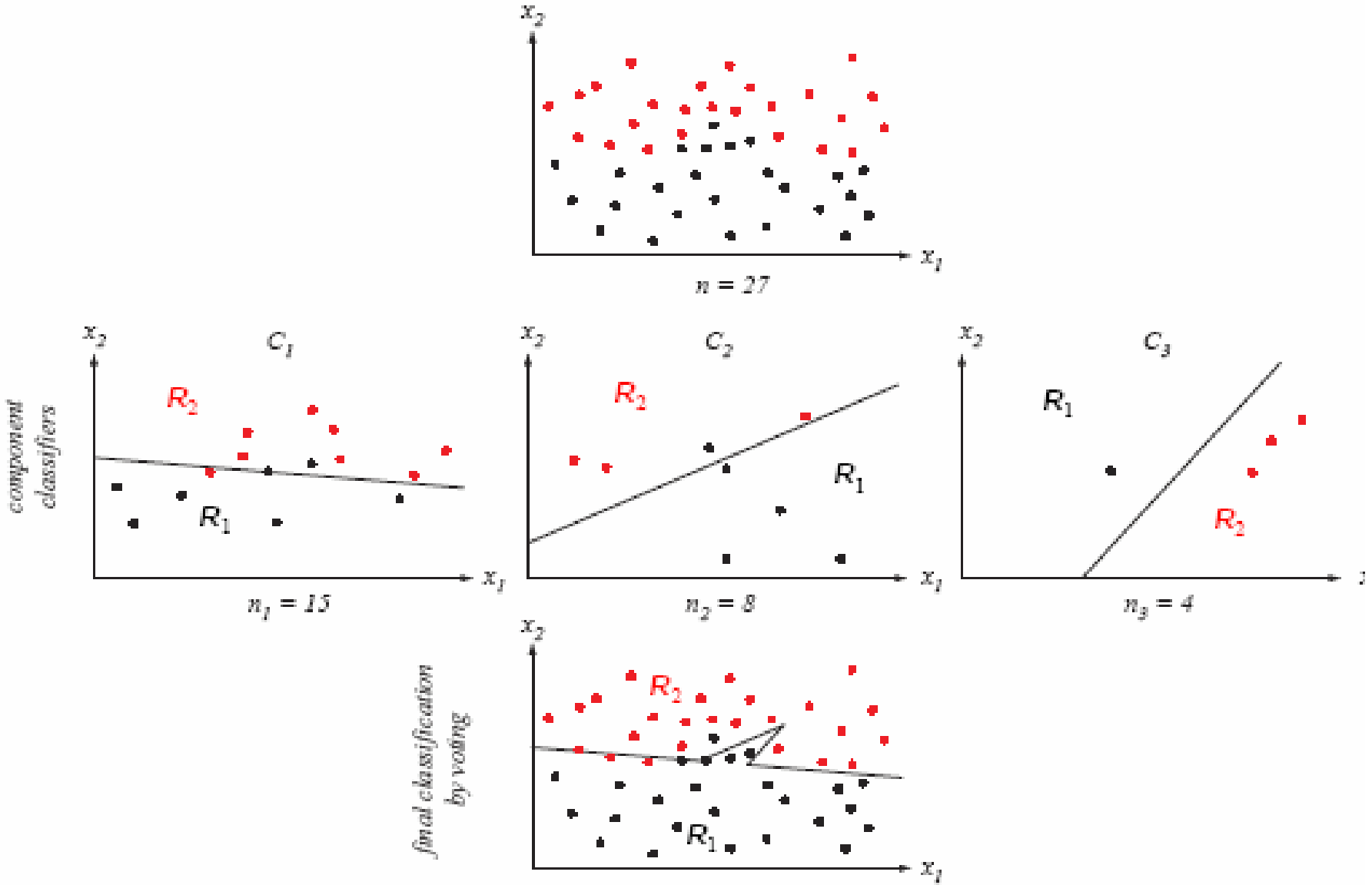
$$\frac{\partial Z_t}{\partial \alpha_t} = 0 \Rightarrow \alpha_t = \frac{1}{2}\log\left(\frac{\sum_{i:h_t(\boldsymbol{x}_i)=y_i} D^t(i)}{\sum_{i:h_t(\boldsymbol{x}_i)\neq y_i} D^t(i)}\right) = \frac{1}{2}\log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

  The resulting upper bound for the training error is

$$\frac{1}{m}\sum_{i=1}^{m}[H(\boldsymbol{x}) \neq y_i] \leq \prod_{t=1}^{T}\left(2\sqrt{\epsilon_t(1-\epsilon_t)}\right)$$

- How to train the weak classifier h_t with a set of weights on the samples?
  – Resample a new training set according to the weights.
  – Use classifiers which can incorporate the weights in the training process (such as a decision tree classifier).

The Basic AdaBoost Algorithm: Example



AdaBoost.M1 (Freund and Schapire, 1995)

1. Given $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_m, y_m)$ where $\boldsymbol{x}_i \in \mathcal{X}$ are samples and $y_i \in \mathcal{Y} = \{1, 2, \cdots, K\}$ are corresponding labels.
2. Initialize sample weights $D^1(i) = \frac{1}{m} \quad (i = 1, 2, \cdots, m)$.
3. For $t = 1, 2, \cdots, T$
   - Train the weak classifier h_t using sample weights $\boldsymbol{D}^t$. $h_t(\mathcal{X}) \rightarrow \mathcal{Y} = \{1, 2, \cdots, K\}$.
   - Choose the weight update parameter $\alpha_t = \frac{1}{2} \log \left(\frac{\sum_{i:h_t(\boldsymbol{x}_i)=y_i} D^t(i)}{\sum_{i:h_t(\boldsymbol{x}_i)\neq y_i} D^t(i)} \right) = \frac{1}{2} \log \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$.
   - Update and normalize the sample weights:

$$D^{t+1}(i) = \frac{D^t(i) \exp\left(-\alpha_t I[h_t(\boldsymbol{x}_i) = y_i]\right)}{Z_t}$$

4. Output the final classifer $H(\boldsymbol{x}) = \arg\max_k \left(\sum_{t=1}^{T} \alpha_t [h_t(\boldsymbol{x}) = k] \right)$

Incorporating Cost-Sensitive Learning

- Cost-sensitive classification considers varying costs of different misclassification types (e.g., missing and false alarm) in the training process.
- Incorporating misclassification costs into the weight update formula:

$$D^{t+1}(i) = \frac{D^t(i)\exp\left(-\alpha_t I[h_t(\boldsymbol{x}_i) = y_i]\right) c_i}{Z_t}$$

  where c_i is the misclassification cost associated with sample $\boldsymbol{x}_i$. Samples with the same class label share the same cost value, i.e., $c_i = c(y_i)$.
- The minority(unbalanced) data set should use higher costs so as to reduce misclassification.

AdaC2.M1 (Main contribution of this paper)

1. Given $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_m, y_m)$ where $\boldsymbol{x}_i \in \mathcal{X}$ are samples and $y_i \in \mathcal{Y} = \{1, 2, \cdots, K\}$ are corresponding labels.
2. Initialize sample weights $D^1(i) = \frac{1}{m} \quad (i = 1, 2, \cdots, m)$.
3. For $t = 1, 2, \cdots, T$
   - Train the weak classifier h_t using sample weights $\boldsymbol{D}^t$. $h_t(\mathcal{X}) \rightarrow \mathcal{Y} = \{1, 2, \cdots, K\}$.
   - Choose the weight update parameter $\alpha_t = \frac{1}{2} \log \left(\frac{\sum_{i:h_t(\boldsymbol{x}_i)=y_i} c_i D^t(i)}{\sum_{i:h_t(\boldsymbol{x}_i)\neq y_i} c_i D^t(i)} \right)$.
   - Update and normalize the sample weights:

$$D^{t+1}(i) = \frac{D^t(i) \exp\left(-\alpha_t I[h_t(\boldsymbol{x}_i) = y_i]\right) c_i}{Z_t}$$

4. Output the final classifer $H(\boldsymbol{x}) = \arg\max_k \left(\sum_{t=1}^{T} \alpha_t [h_t(\boldsymbol{x}) = k] \right)$

Incorporating Cost-Sensitive Learning

- The cost vector for K classes $[c(1), c(2), \cdots, c(K)]$ should be determined before using the AdaC2.M1 algorithm.
- The paper proposed a Genetic Algorithm to search for the optimal cost setup.
- The cost vector is encoded as a binary string. Then candidate cost vectors are evaluated by the performance of the AdaC2.M1 algorithm.
- Finally he fittest cost vector will survive and be used in the training process afterwards.

Experimental Results: Performance Measures

- The confusion matrix:

Class labels	1	2	3	$\cdots$	$K-1$	K
1	n_{11}	n_{12}	n_{13}	$\cdots$	$n_{1,K-1}$	$n_{1,K}$
2	n_{21}	n_{22}	n_{23}	$\cdots$	$n_{2,K-1}$	$n_{2,K}$
3	n_{31}	n_{32}	n_{33}	$\cdots$	$n_{3,K-1}$	$n_{3,K}$
$\cdots$	.	.	.	$\cdots$	.	.
K	$n_{K,1}$	$n_{K,2}$	$n_{K,3}$	$\cdots$	$n_{K,K-1}$	$n_{K,K}$

n_{ij} is the number of samples whose real label is i but classified as j.

- Accuracy: $A = \frac{\sum_{i=1}^{K} n_{ii}}{\sum_{i=1}^{K}\sum_{j=1}^{K} n_{ij}}$. Recall: $R_i = \frac{n_{ii}}{\sum_{j=1}^{K} n_{ij}}$.

  Precision: $R_i = \frac{n_{ii}}{\sum_{j=1}^{K} n_{ji}}$. G-mean: G-mean $= \left(\prod_{i=1}^{K} R_i\right)^{\frac{1}{K}}$

Experimental Results: Setup

- Used Car data in UCI Machine Learning Database for car evaluation. There are 1728 instances and are grouped into 4 classes. Each sample is described by 6 nominal ordered attributes.
- The data set is used for both cost setup searching and performance evaluation.
- For performance evaluation, 80% of the data is used as training set and the remaining 20% as testing set.
- Used a decision tree classification system (C4.5) as the weak classifier.
- The cost vector is found in advance using the Genetic Algorithm.

Experimental Results: Results

Table 2. Class Distribution

index	class name	class size	class distribution
C1	unacc	1210	70.023%
C2	acc	384	22.222%
C3	good	69	3.993%
C4	v-good	65	3.762%

Table 4. G-mean Evaluation

Class	C4.5	AdaBoost.M1	AdaC2.M1
C1(R)	0.9637	0.9707	0.9586
C2(R)	0.9083	0.9056	0.9459
C3(R)	0.7175	0.7395	0.8540
C4(R)	0.7902	0.9151	0.9139
G-mean	0.8336	0.8758	0.9146

Cost vector: $[0.3281, 0.6682, 0.7849, 1.0000]$

Conclusions

- AdaBoost deals with the class imbalance problem by maintaining a set of weights on the training data set in the learning process.
- Cost-sensitive learning can also be incorporated into the weight update, forcing the classifier to focus on the minority data set.
- Extension to multi-class problems is straightforward, under the condition that the weak classifier is for multi-class data set.
- Satisfactory results are obtained using the proposed AdaC2.M1 algorithm on an unbalanced data set.