12. Ensemble Methods: Bagging and Boosting

Ensemble Methods

Why?

Some machine learning models such as decision trees are simple but often overfit leading to weak noisy predictors.

Ensemble methods combine several machine learning techniques into one predictive model in order to decrease variance and/or bias. Random forests are examples of ensemble methods.

Examples

Bagging

Bagging can fit many large trees to bootstrap-resampled versions of the training data.

Classification is done via majority vote.

Boosting

Boosting can fit many large or small trees to reweighted versions of the training data.

Classification is done via weighted majority vote.

Bagging: Overview

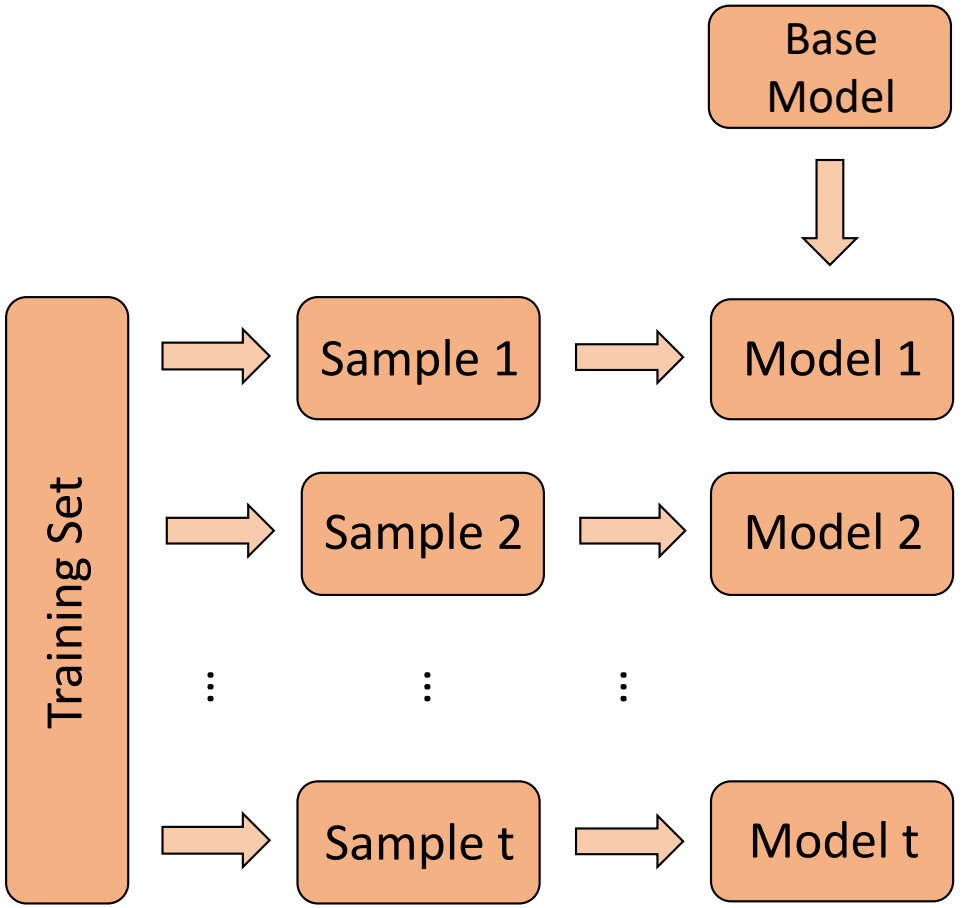
Overview

Bagging over a given ‘base’ procedure over many samples to improve its performance

It requires a class of ‘base’ learning models such as decision trees, neural networks, etc.

It also uses a training set consisting of many examples to train various models leading to an aggregate overall model.

It then makes predictions on a testing set based on the aggregate overall model.



Bagging: Overview

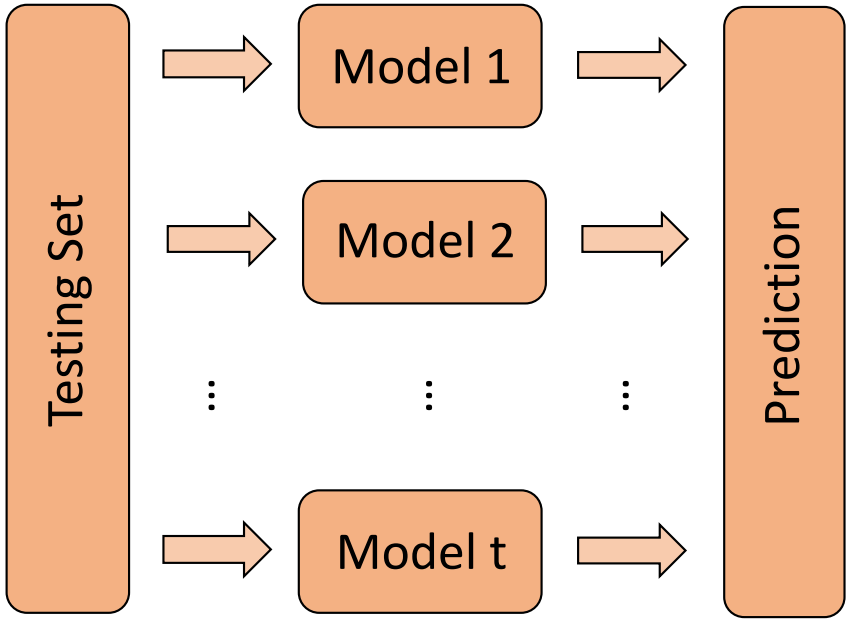
Overview

Bagging over a given ‘base’ procedure over many samples to improve its performance

It requires a class of ‘base’ learning models such as decision trees, neural networks, etc.

It also uses a training set consisting of many examples to train various models leading to an aggregate overall model.

It then makes predictions on a testing set based on the aggregate overall model.



Bagging: Procedure

Procedure

Training

For each ‘iteration’

(1) Randomly sample with replacement a number of samples from the training set

(2) Fit a chosen ‘base model’ to the samples (e.g. a neural network or decision tree)

Testing

For each test example, the prediction is carried out by combining the predictions of the various trained models e.g. by averaging (regression) or majority vote (classification)

Bagging: Procedure

Example

input:

Decision Tree T_S ,training set $\mathcal{S}$, test data point to be labelled x.

procedure:

Draw bootstrap samples $\mathcal{S}_1^*, \mathcal{S}_2^*, \ldots, \mathcal{S}_B^*$, of a certain size with replacement from $\mathcal{S}$

Fit decision trees $T_{\mathcal{S}_1^*}, T_{\mathcal{S}_2^*}, \ldots, T_{\mathcal{S}_B^*}$ to the samples $\mathcal{S}_1^*, \mathcal{S}_2^*, \ldots, \mathcal{S}_B^*$

output:

For regression:

return $T_{bag}(x) = {}^1/_B \sum_{b=1}^{B} T_{\mathcal{S}_b^*}(x)$

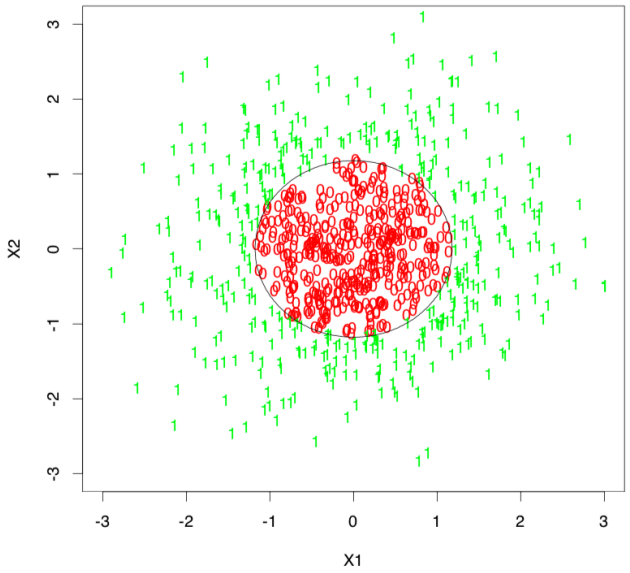
For classification:

return $T_{bag}(x) = Majority\ Vote\left\{T_{\mathcal{S}_b^*}(x)\right\}_{b=1}^{B}$

Bagging: Illustration

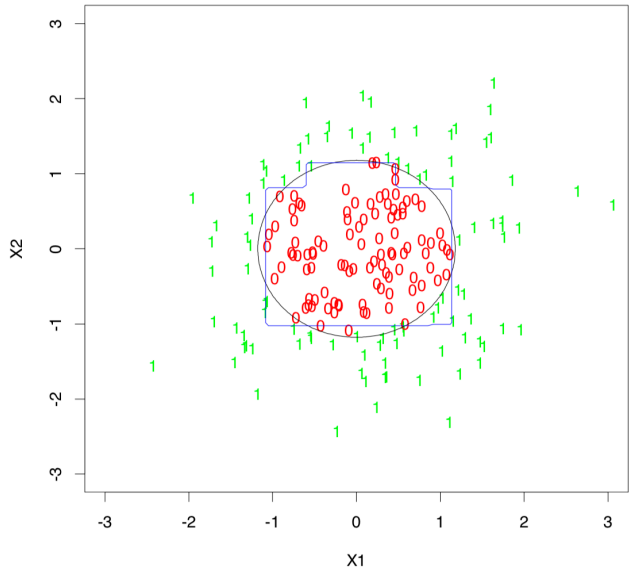
Illustration

Bayes Error = 0



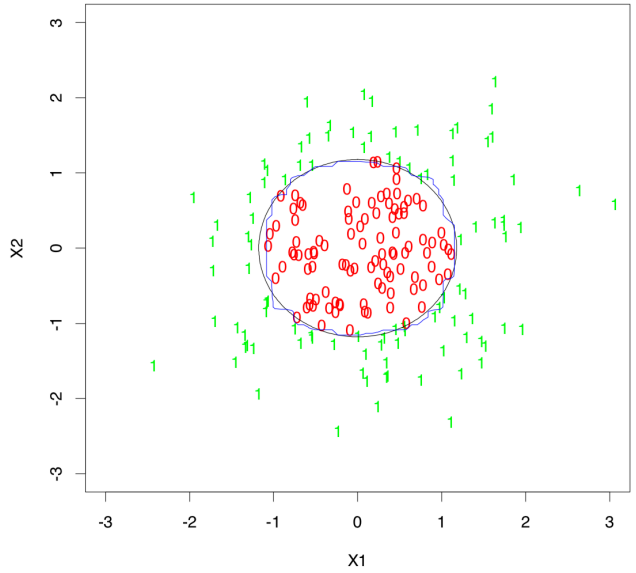
Dataset

Error = 0.073



Single Decision Tree
Non-smooth decision boundaries

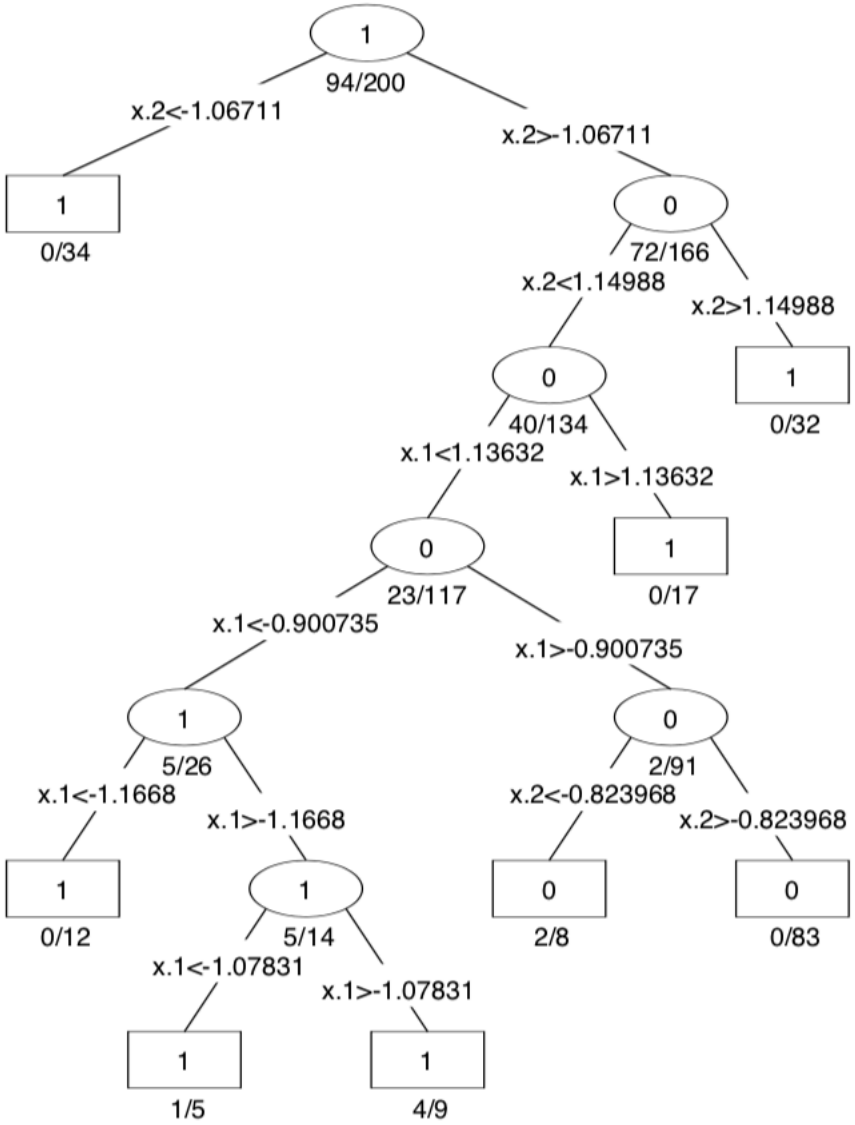
Error = 0.032



Bagging
Smooth decision boundaries

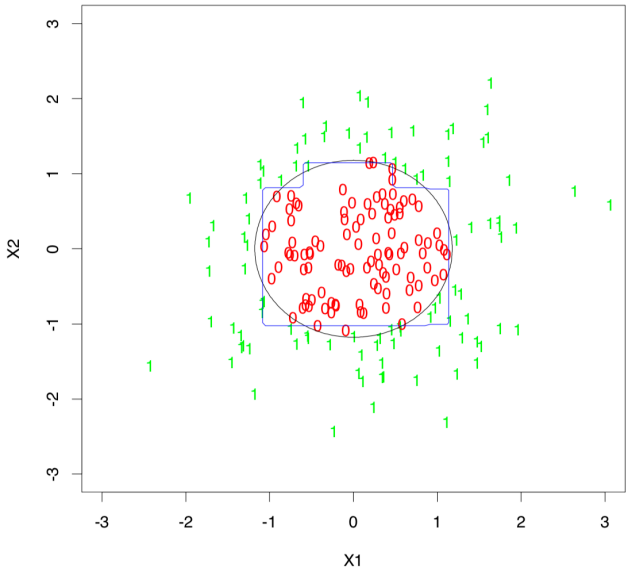
Adapted from Trevor Hastie, Stanford University

Decision Tree



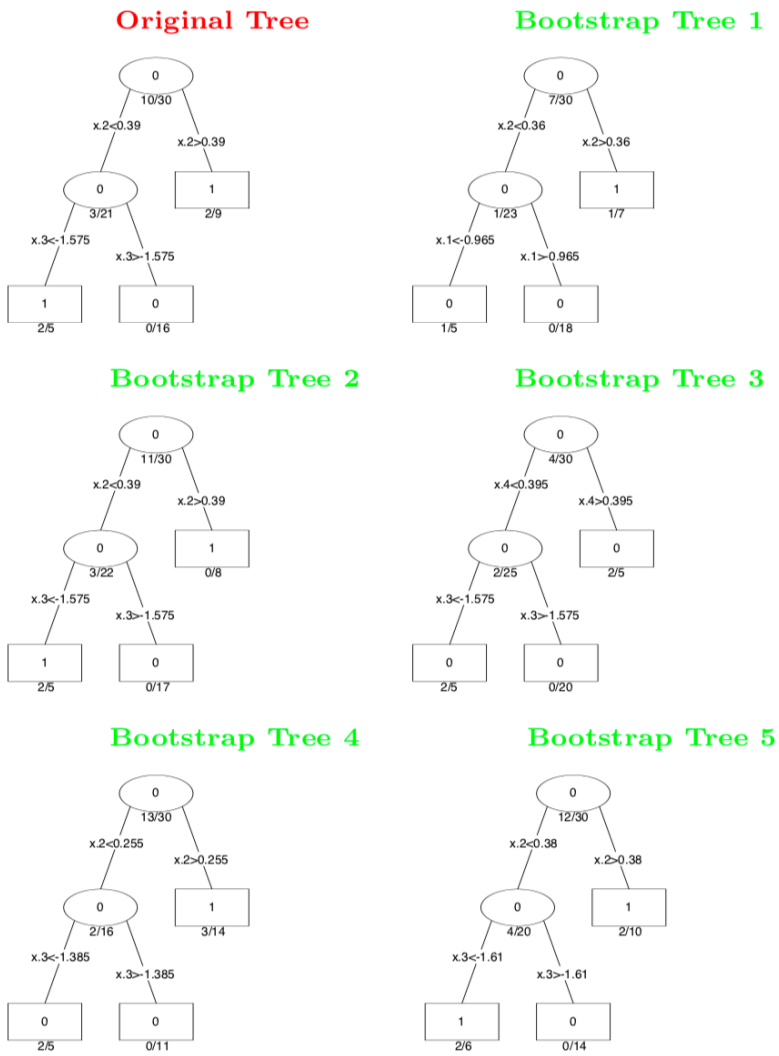
Adapted from Trevor Hastie, Stanford University

Error = 0.073

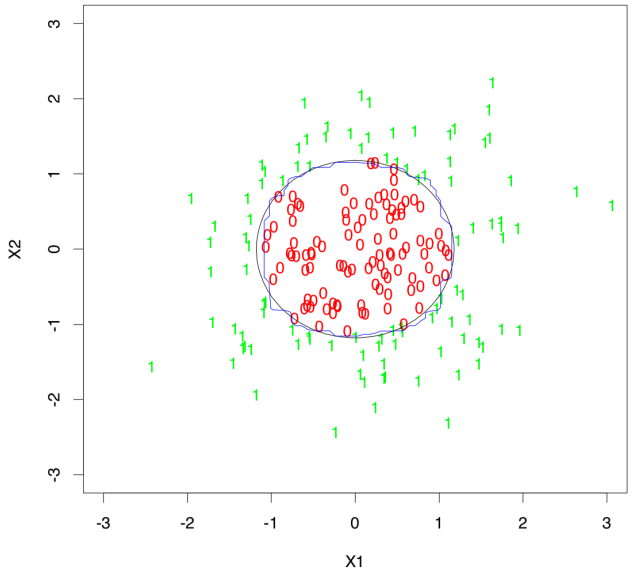


Single Decision Tree
Non-smooth decision boundaries

Bagging



Error = 0.032



Bagging
Smooth decision boundaries

Adapted from Trevor Hastie, Stanford University

Bagging: Considerations

Considerations

Under-fitting is associated with high-bias models – models may not fit well the training data – leading to small variance (in the sense that training is less susceptible to the influence of examples in the training set)

Over-fitting is associated with low-bias models – models may overfit the training data – leading to high variance (in the sense that training is more susceptible to the influence of examples in the training set)

Bagging can help by decreasing variance of the base model without influencing bias, but can also result in loss of simple structures in the original model.

Boosting: Motivation

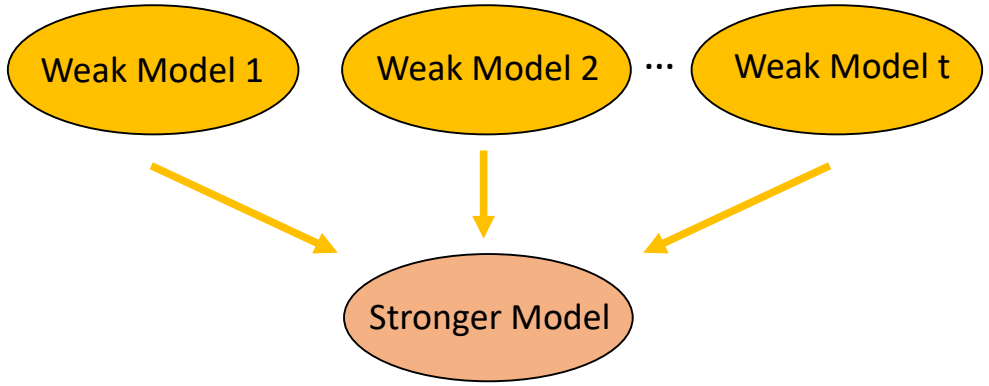
Motivation

Strong learners lead to models that make arbitrarily accurate predictions most of the time.

Weak learners lead to models that make predictions that are only slightly better than random guesses.

Boosting has been motivated by the question whether one can convert an ensemble of weak models onto a strong one?

This can give rise to computational efficiency gains – but not statistical gains – provided that weaker models are easier to learn than stronger ones



A combination of weaker models leading to a stronger one

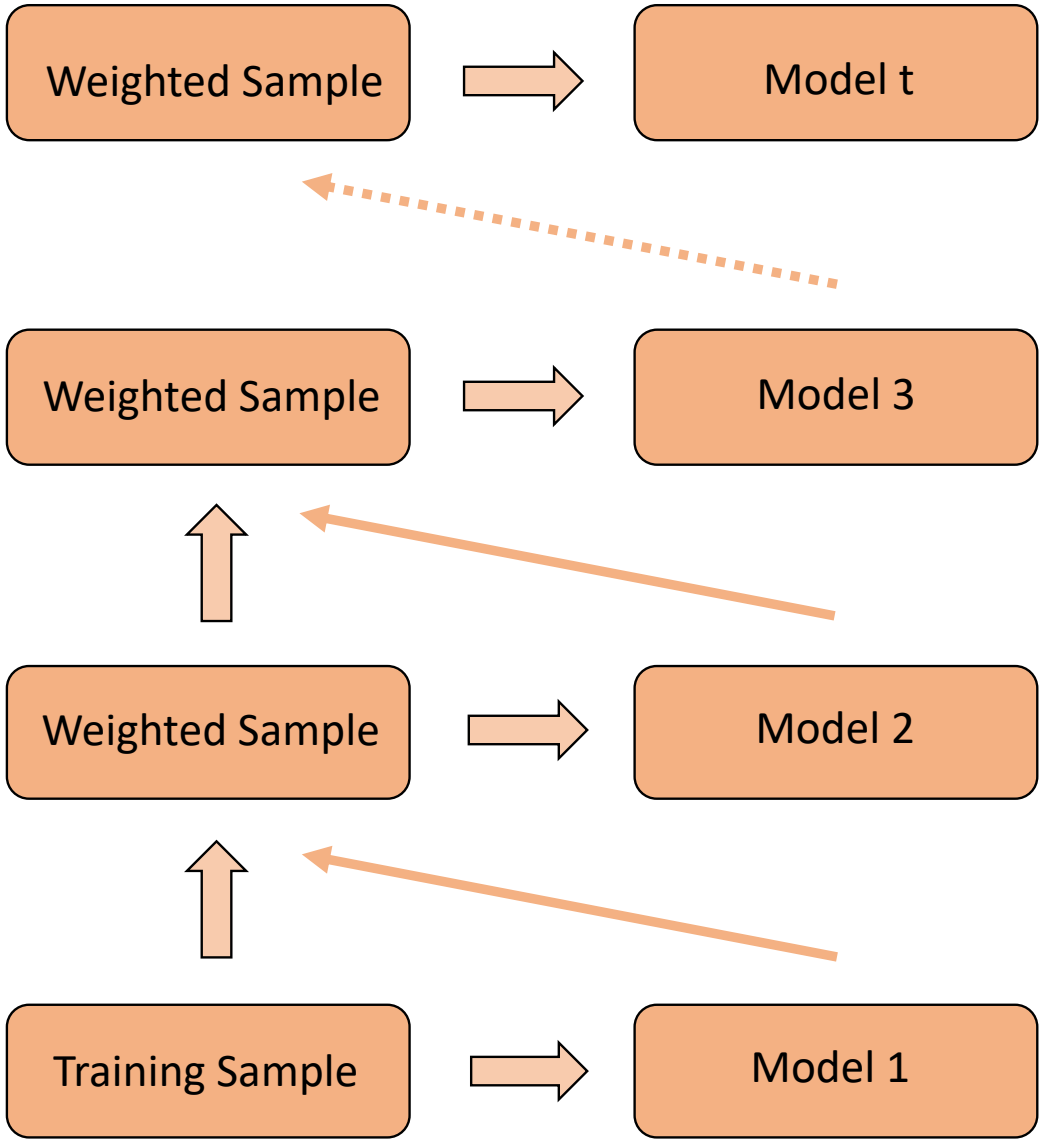
Boosting: Overview

Overview

Boosting is one of the most successful machine learning ideas that can be used to improve the performance of any supervised learning algorithm

It sequentially learns an ensemble of weaker models using weighted versions of the samples so that each model tries to correct mistakes of previous models

It then combines the weaker models onto a stronger one



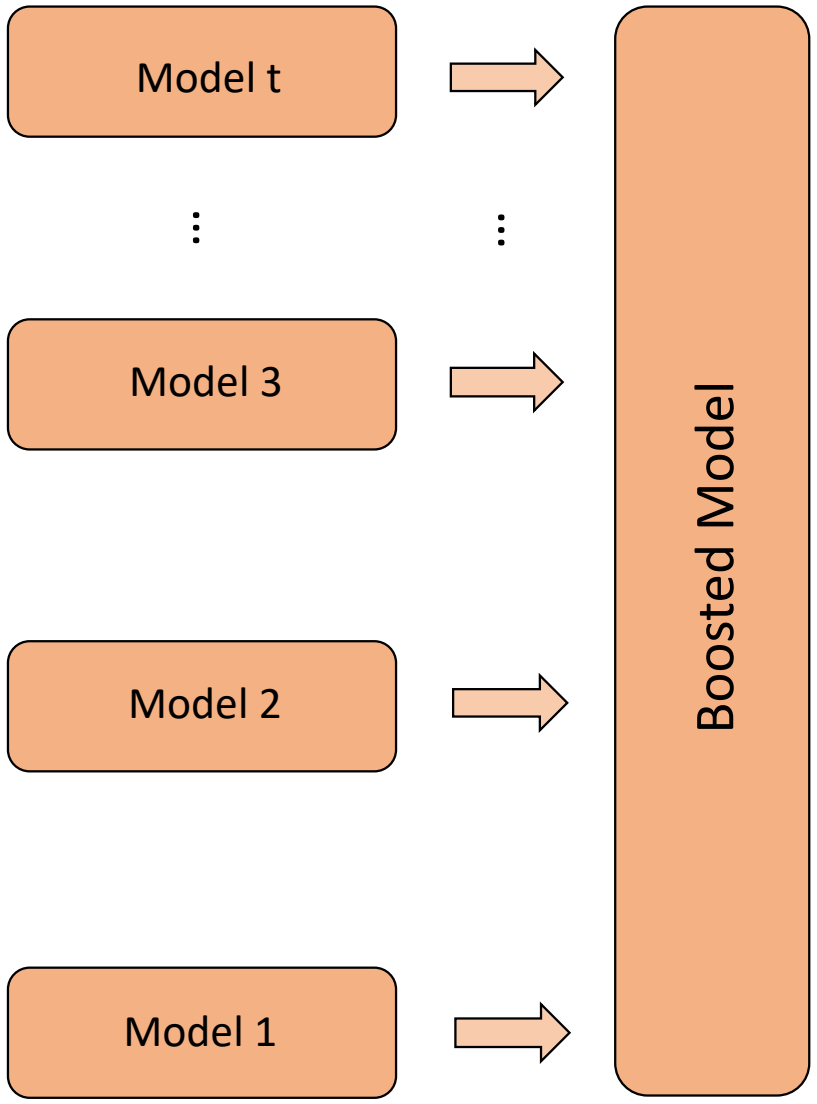
Boosting: Overview

Overview

Boosting is one of the most successful machine learning ideas that can be used to improve the performance of any supervised learning algorithm

It sequentially learns an ensemble of weaker models using weighted versions of the samples so that each model tries to correct mistakes of previous nmodels

It then combines the weaker models onto a stronger one



Boosting: Procedure

Procedure

Training

We start by assigning weights $w_i^1 = 1/n$ to the various points in the training set $\{(x_i, y_i); i = 1, \dots, n\}$

We then iterate as follows:

(1) At iteration b $(b = 1, \dots, B)$, we train a weak classifier $y^b(x)$ on a weighted training set $\{(x_i, y_i, w_i^b); i = 1, \dots, n\}$

(2) At iteration b $(b = 1, \dots, B)$, we also update the weights in a way such that weights are increased for all points that are wrongly classified by the weak classifier $y^b(x)$ and weights are decreased for all points that are correctly classified by the classifier $y^b(x)$

Testing

We make a prediction $y_{boost}(x)$ for a data point x by combining the predictions of the various classifiers $y^1(x), y^2(x), \dots, y^B(x)$. For example, in binary classification where $y \in \{-1, +1\}$, the prediction is given by

$$y_{boost}(x) = sign\left(\sum_{b=1}^{B} \alpha^b y^b(x)\right)$$

Adaboost

Procedure (Binary Classification)

We start by assigning weights $w_i^1 = 1/n$ to the various points in the training set $\{(x_i, y_i); i = 1, \dots, n\}$

We then iterate as follows:

(1) At iteration b $(b = 1, \dots, B)$, we train a weak classifier $y^b(x)$ on a weighted training set $\{(x_i, y_i, w_i^b); i = 1, \dots, n\}$

(2) At iteration b $(b = 1, \dots, B)$, we also update the weights as follows:

(a) Compute $E_{train}^b = \sum_{i=1}^n w_i^b 1\{y_i \neq y^b(x)\}$

(b) Compute $\alpha^b = 0.5 \log\left((1 - E_{train}^b)/E_{train}^b\right)$

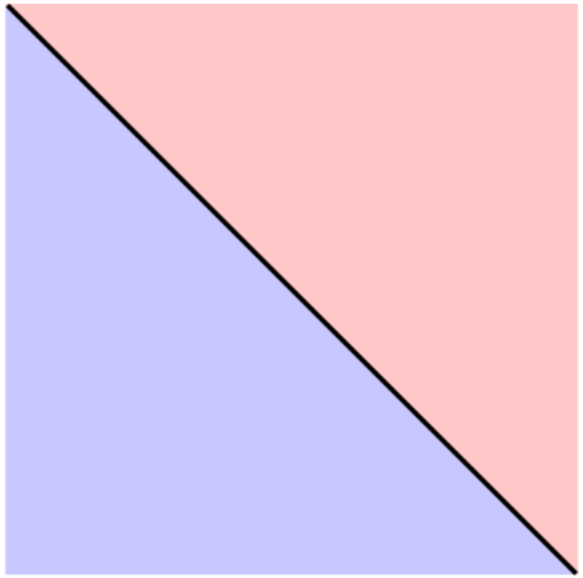
(c) Compute $w_i^{b+1} = w_i^b exp\left(-\alpha^b y_i y^b(x)\right), i = 1, \dots, n$

(d) Set $w_i^{b+1} \leftarrow w_i^{b+1} / \sum_{j=1}^n w_j^{b+1}, i = 1, \dots, n$

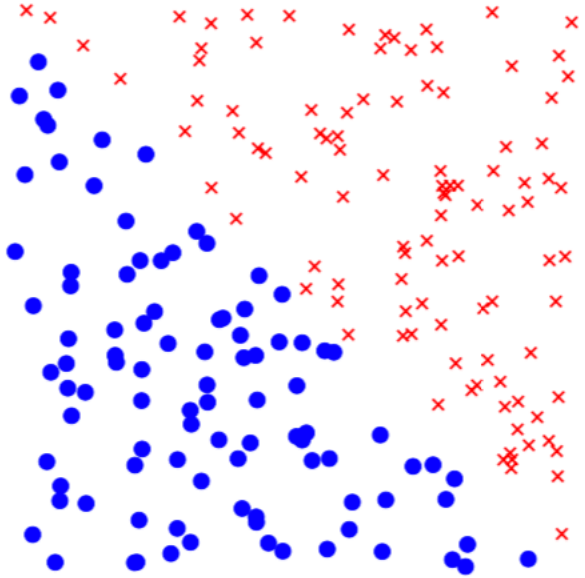
(3) We finally output $y_{boost}(x) = sign\left(\sum_{b=1}^B \alpha^b y^b(x)\right)$

NB: Each successive classifier is forced to concentrate on samples missed by previous classifiers.

Adaboost: Illustration



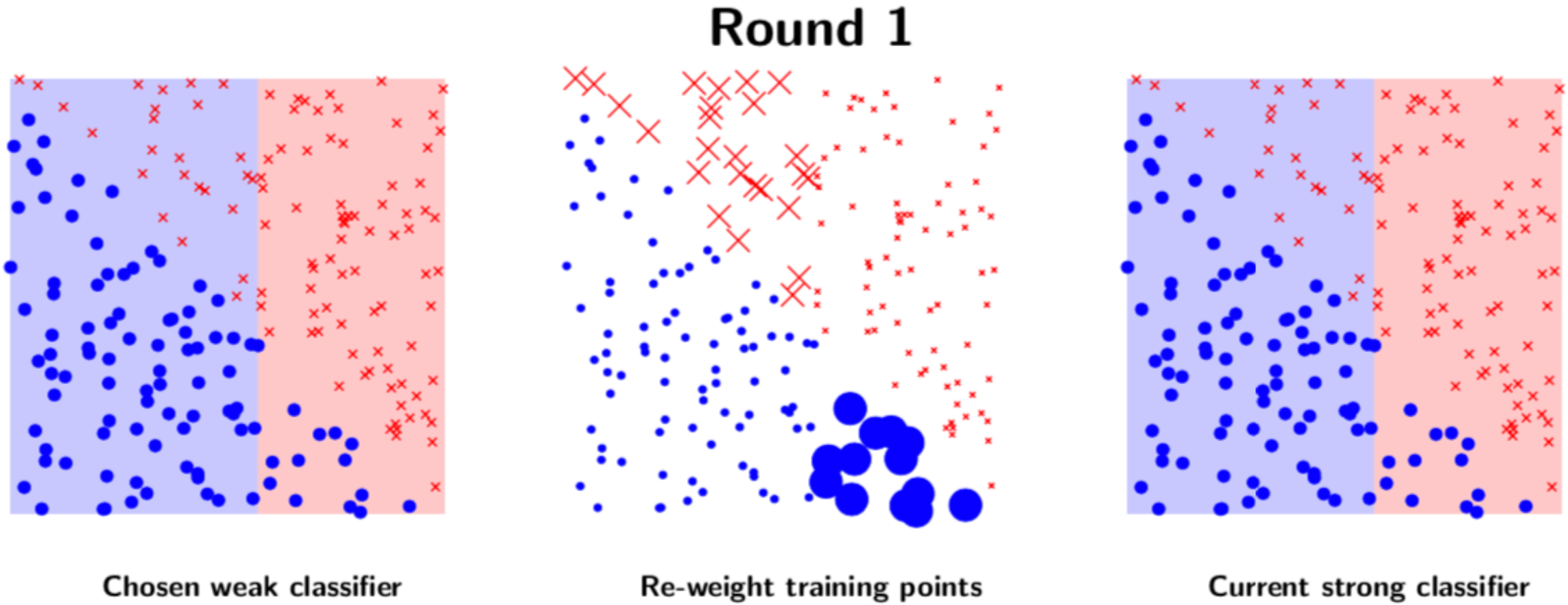
True decision boundary



Training data

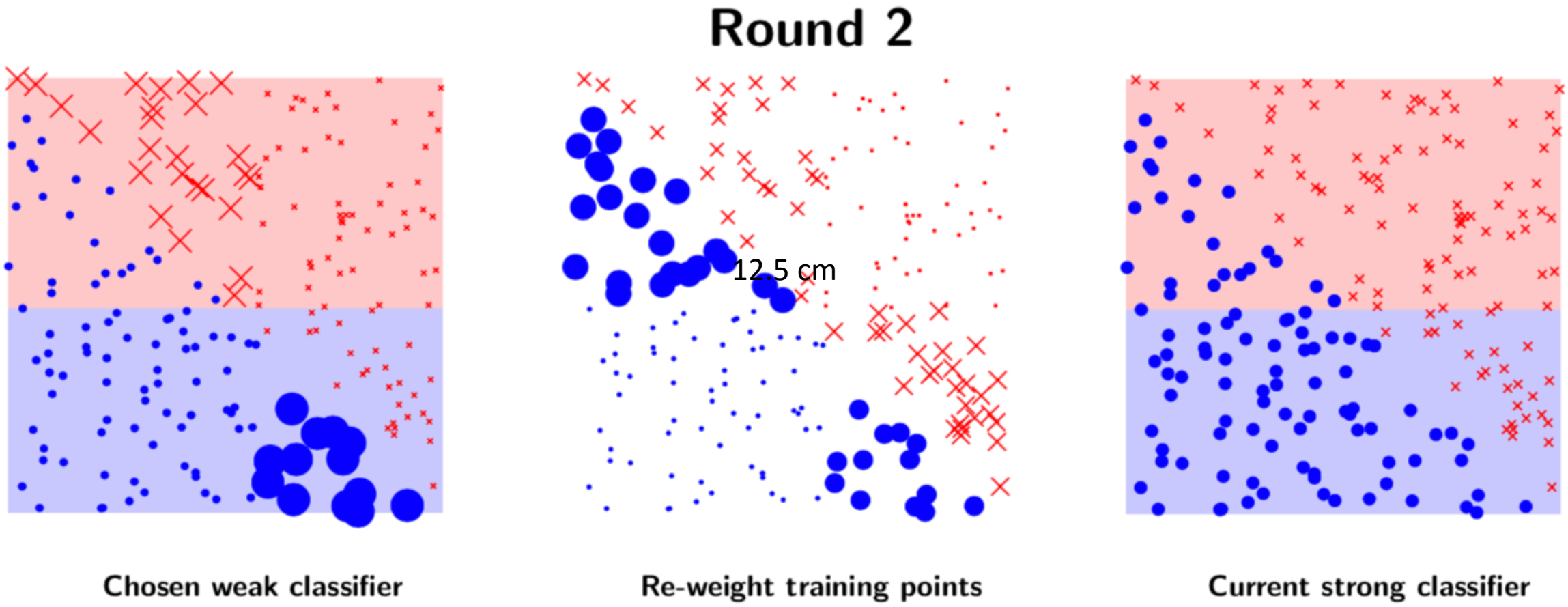
Source: DD3364, KTH

Adaboost: Illustration



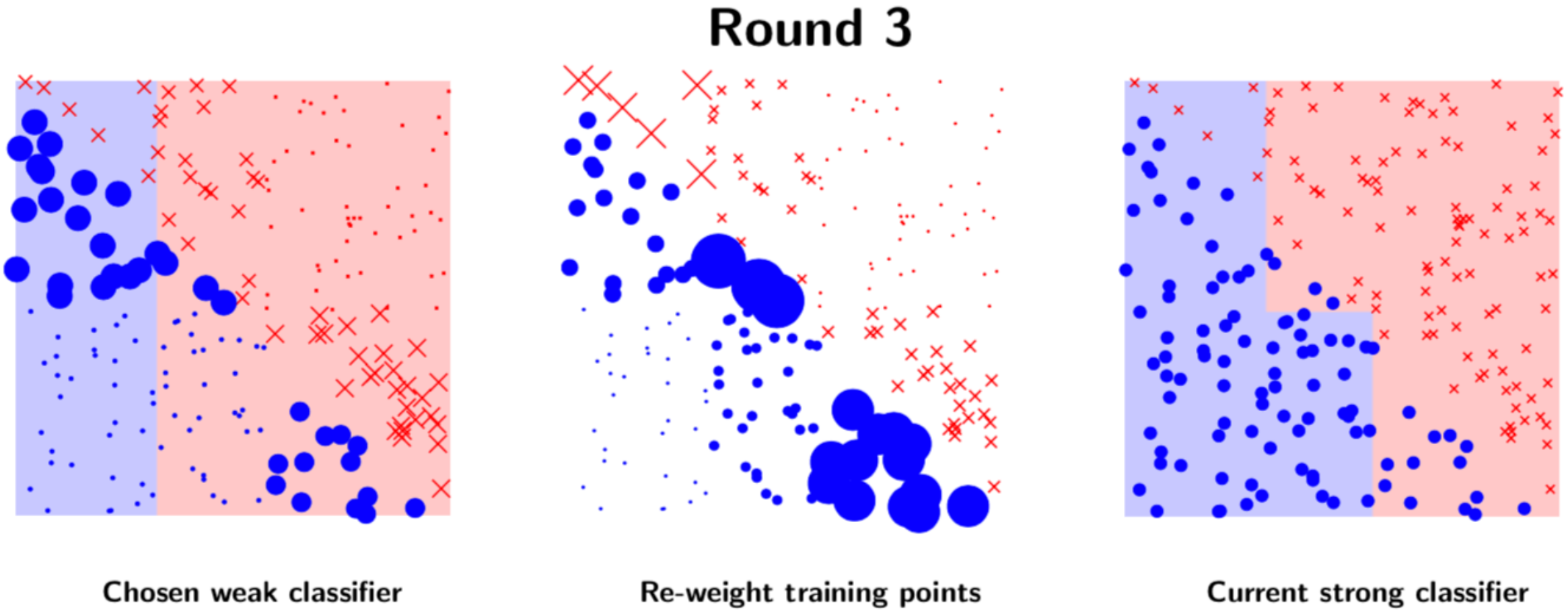
Source: DD3364, KTH

Adaboost: Illustration



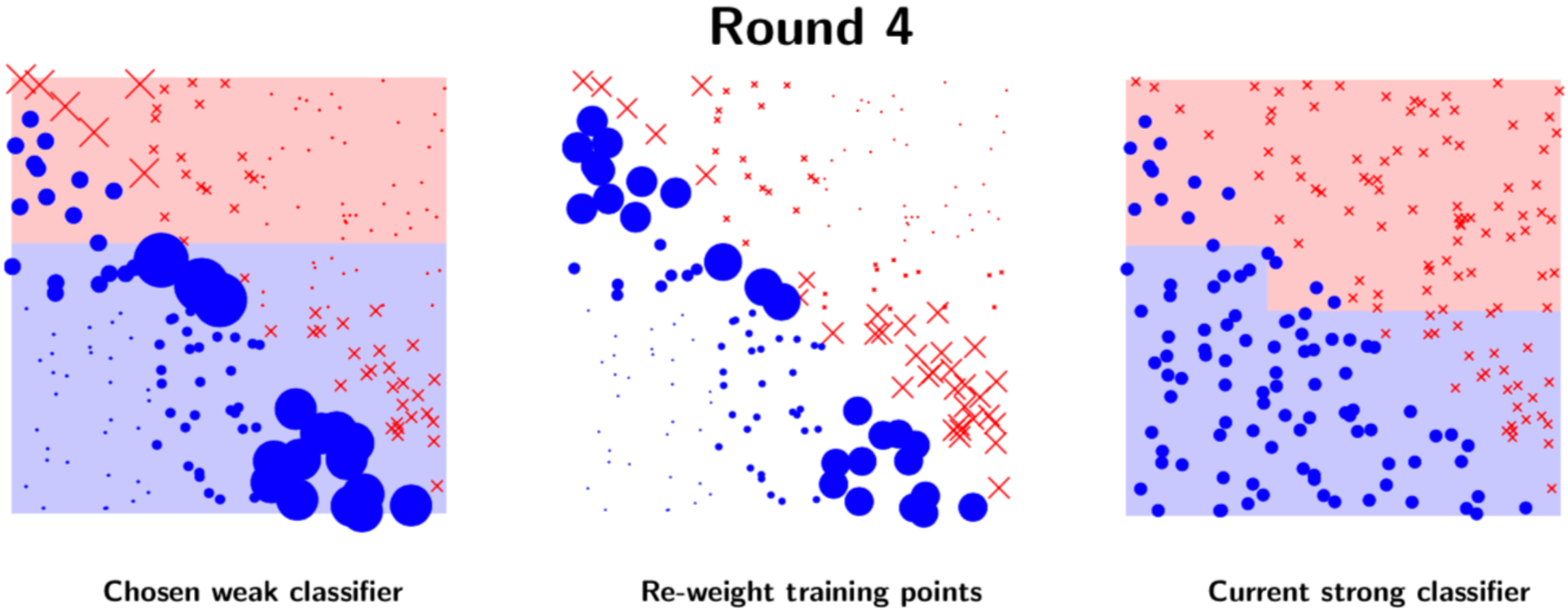
Source: DD3364, KTH

Adaboost: Illustration



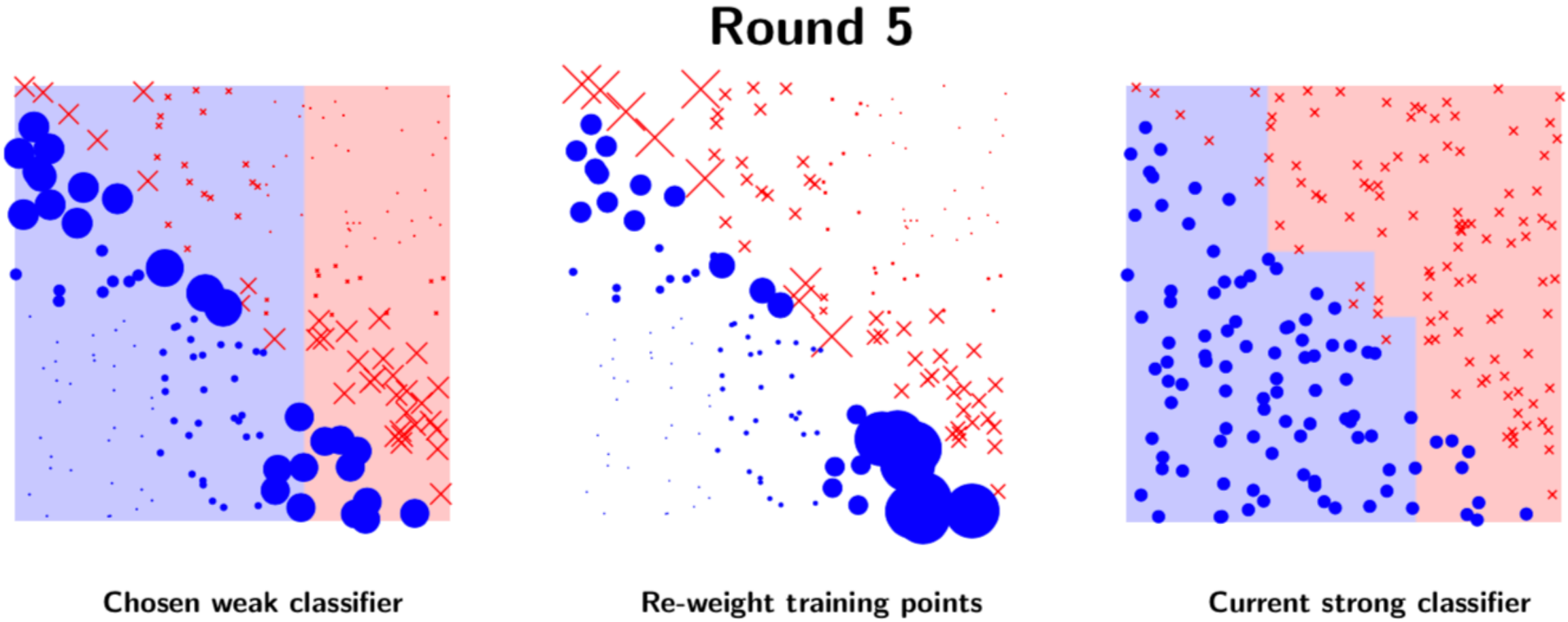
Source: DD3364, KTH

Adaboost: Illustration



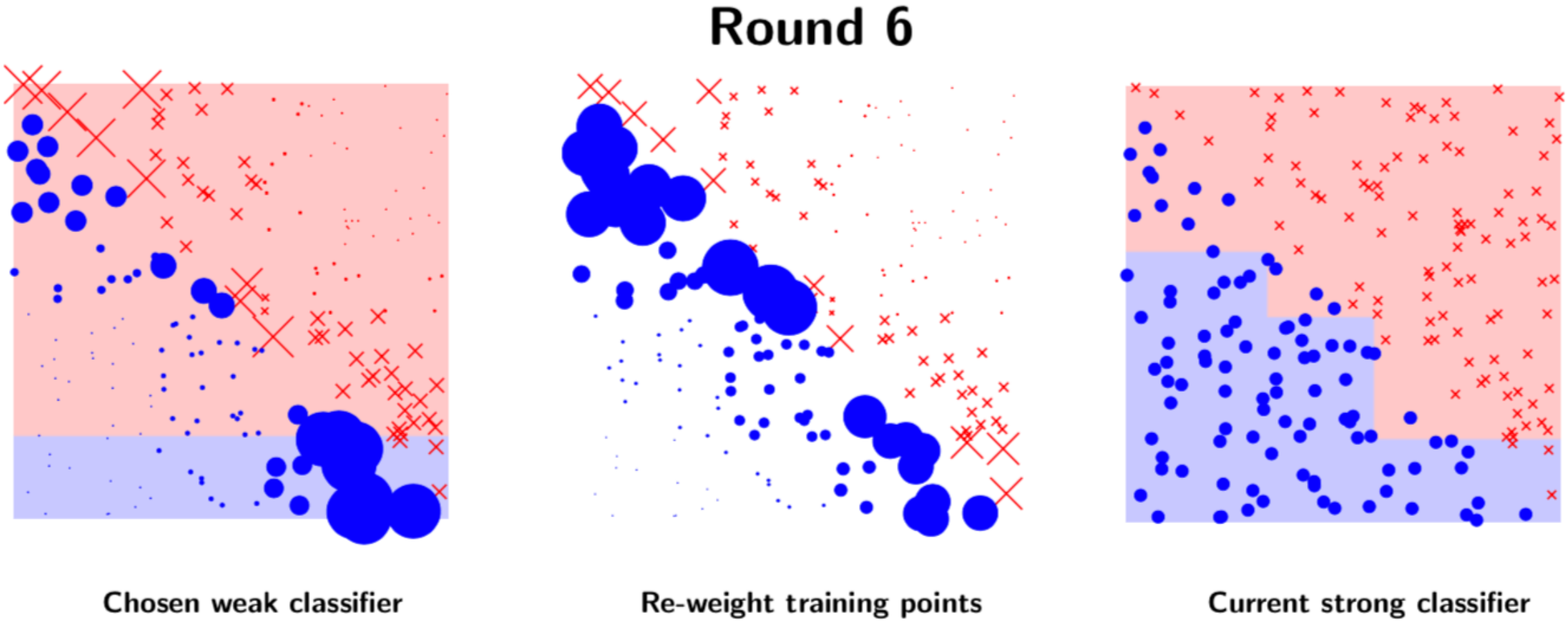
Source: DD3364, KTH

Adaboost: Illustration



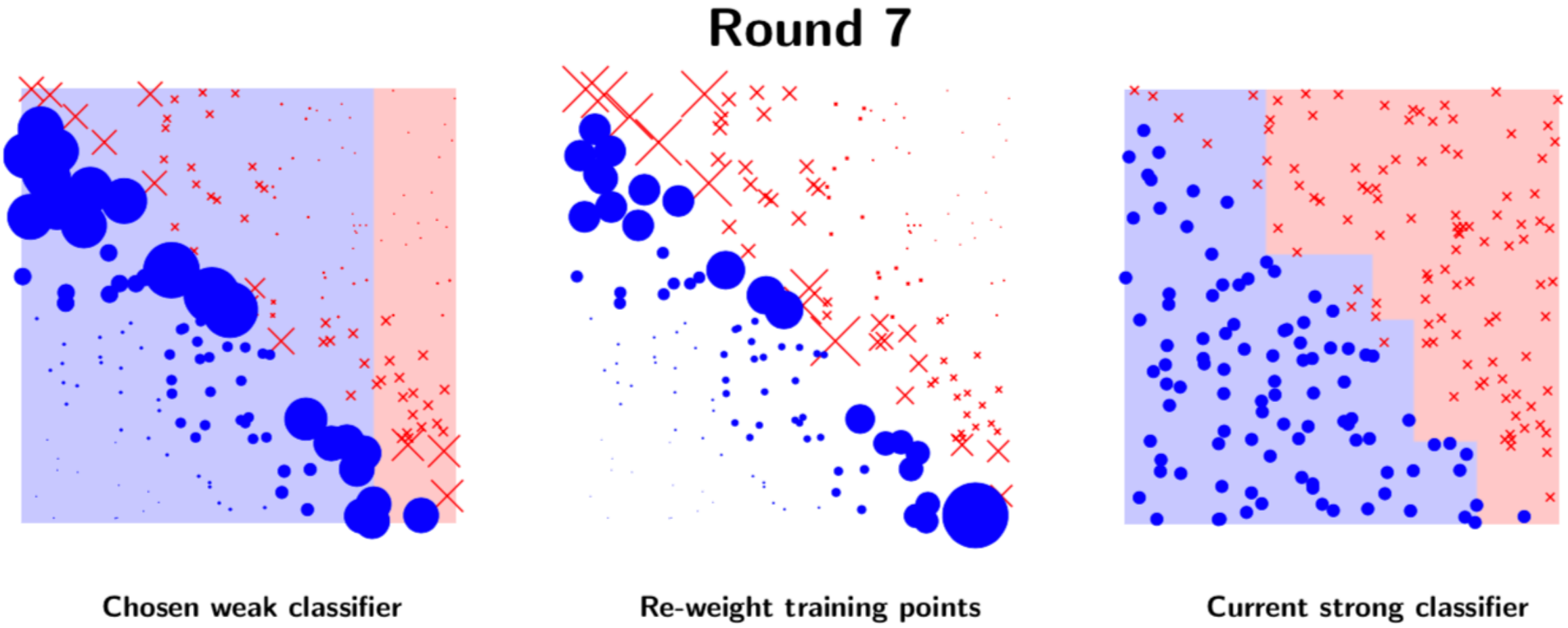
Source: DD3364, KTH

Adaboost: Illustration



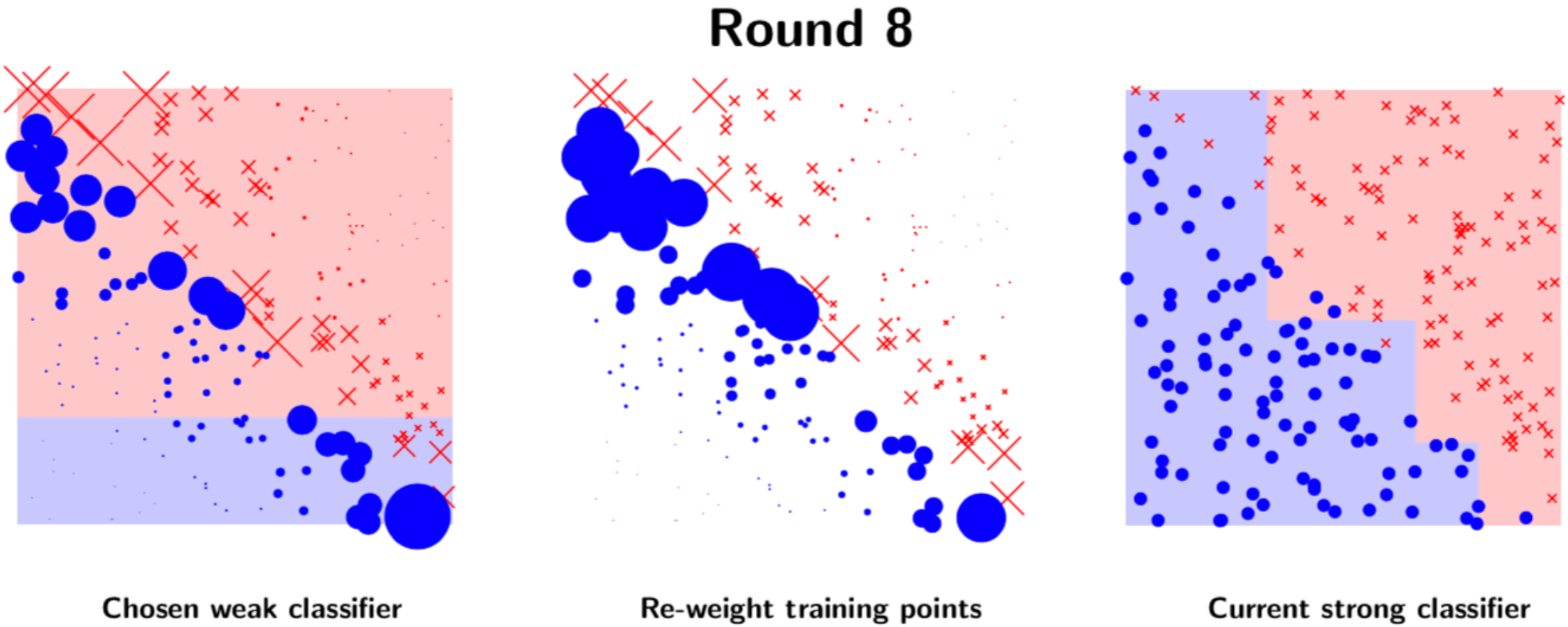
Source: DD3364, KTH

Adaboost: Illustration



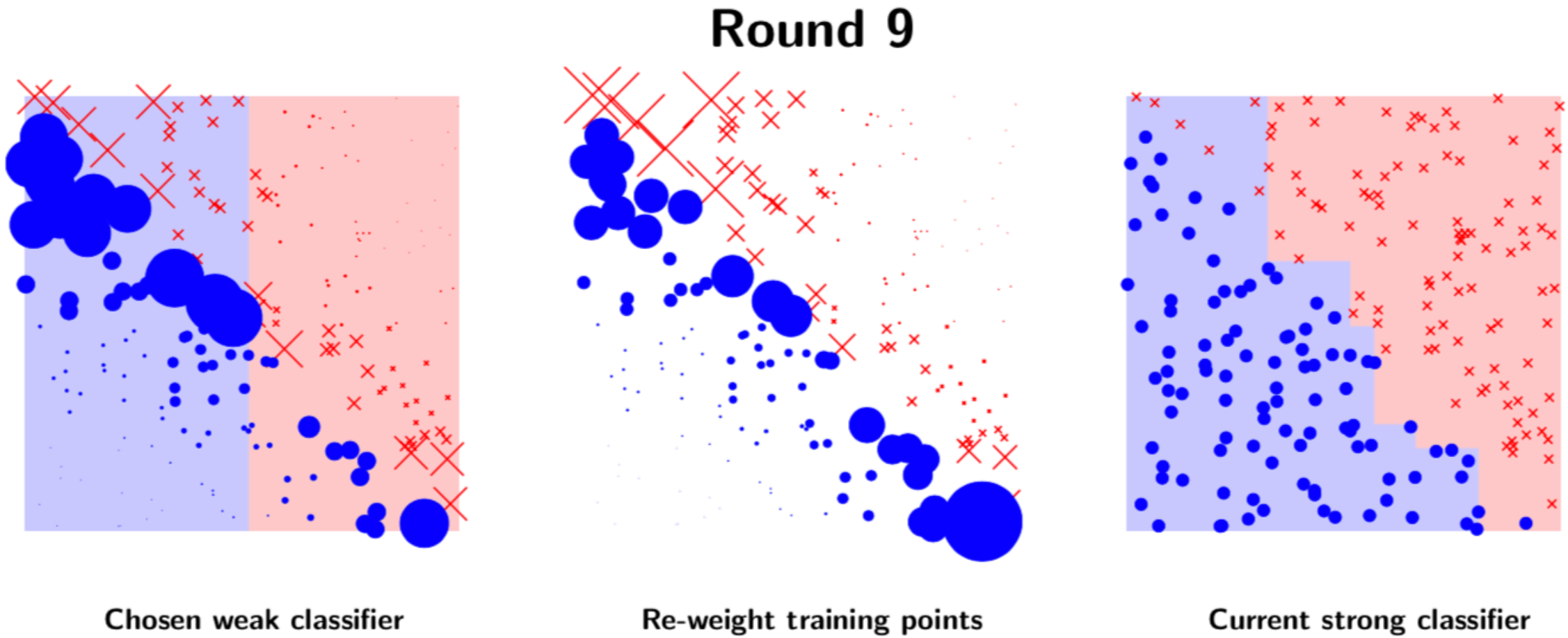
Source: DD3364, KTH

Adaboost: Illustration



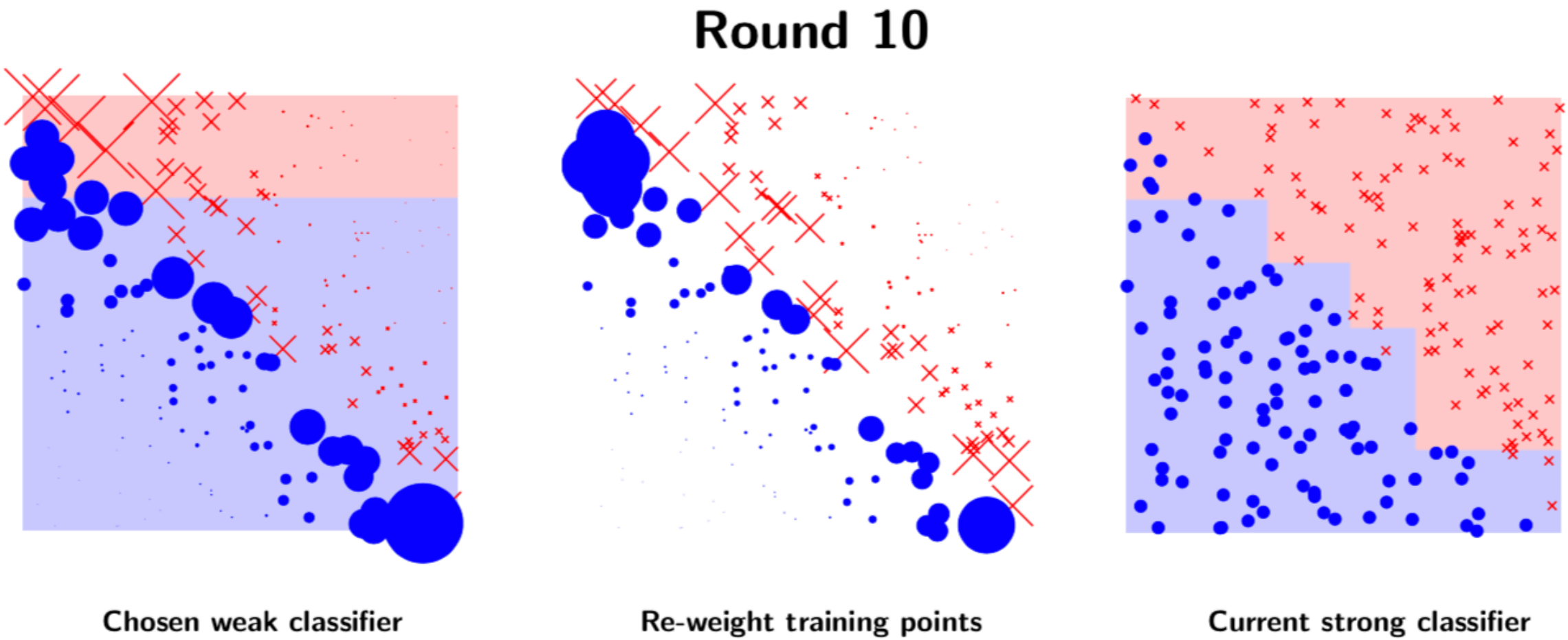
Source: DD3364, KTH

Adaboost: Illustration



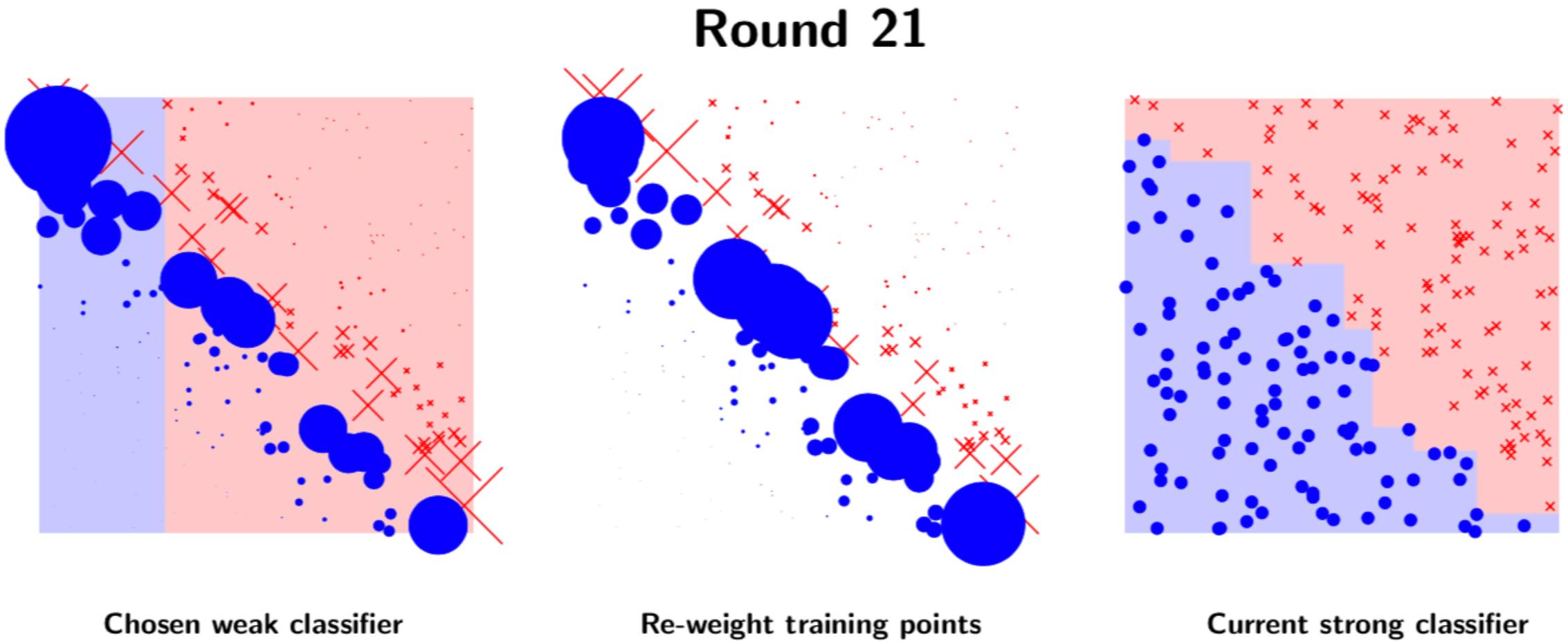
Source: DD3364, KTH

Adaboost: Illustration



Source: DD3364, KTH

etc



Source: DD3364, KTH

Adaboost: Interpretation

Interpretation

One is fitting an additive expansion in a set of elementary functions as follows:

$$y(x) = sign\left(\sum_{b=1}^{B} \alpha^b y^b(x)\right)$$

where the basis functions $y^b(x) \in \{-1, +1\}$ are weak classifiers.

Generally, basis function expansions can be expressed as follows:

$$f(x) = \sum_{b=1}^{B} \alpha_b b(x; \beta_b)$$

where α_b are the expansions coefficients and $b(x; \beta_b)$ are elementary functions parameterized by β_b

Adaboost: Interpretation

Interpretation

A learning approach would involve solving the optimization problem given by:

$$\min_{\alpha_1,\beta_!,\ldots,\alpha_B,\beta_B} \sum_{i=1}^{n} l\left(y_i, \sum_{b=1}^{B} \alpha_b b(x_i;\beta_b)\right)$$

where $l(\cdot,\cdot)$ is a loss function.

This can be a very hard optimization problem!

Adaboost: Interpretation

Forward Stagewise Additive Modelling

Another learning approach would involve greedily adding one basis function at a time as follows:

(1) Set $\mathrm{f}_0(x) = 0$

(2) For $b = 1, \ldots, B$

(a) Set

$$\hat{\alpha}_b, \hat{\beta}_b = \underset{\alpha_b, \beta_b}{\operatorname{argmin}} \sum_{i=1}^{n} l\big(y_i, f_b(x) + \alpha_b b(x; \beta_b)\big)$$

(b) Set

$$f_b(x) = f_{b-1}(x) + \hat{\alpha}_b b\big(x; \hat{\beta}_b\big)$$

Adaboost (M1) implements forward stage modelling with an exponential loss function given by:

$$l\big(y, f(x)\big) = exp\big(-y \cdot f(x)\big)$$

Adaboost: Considerations

Considerations

The approximation error decreases exponentially with the number of iterations.

The estimation error in contrast increases with the number of iterations.

Therefore, one can use the number of iterations to control the bias-variance tradeoff; in particular, the number of iterations can be optimized using cross-validation procedures.

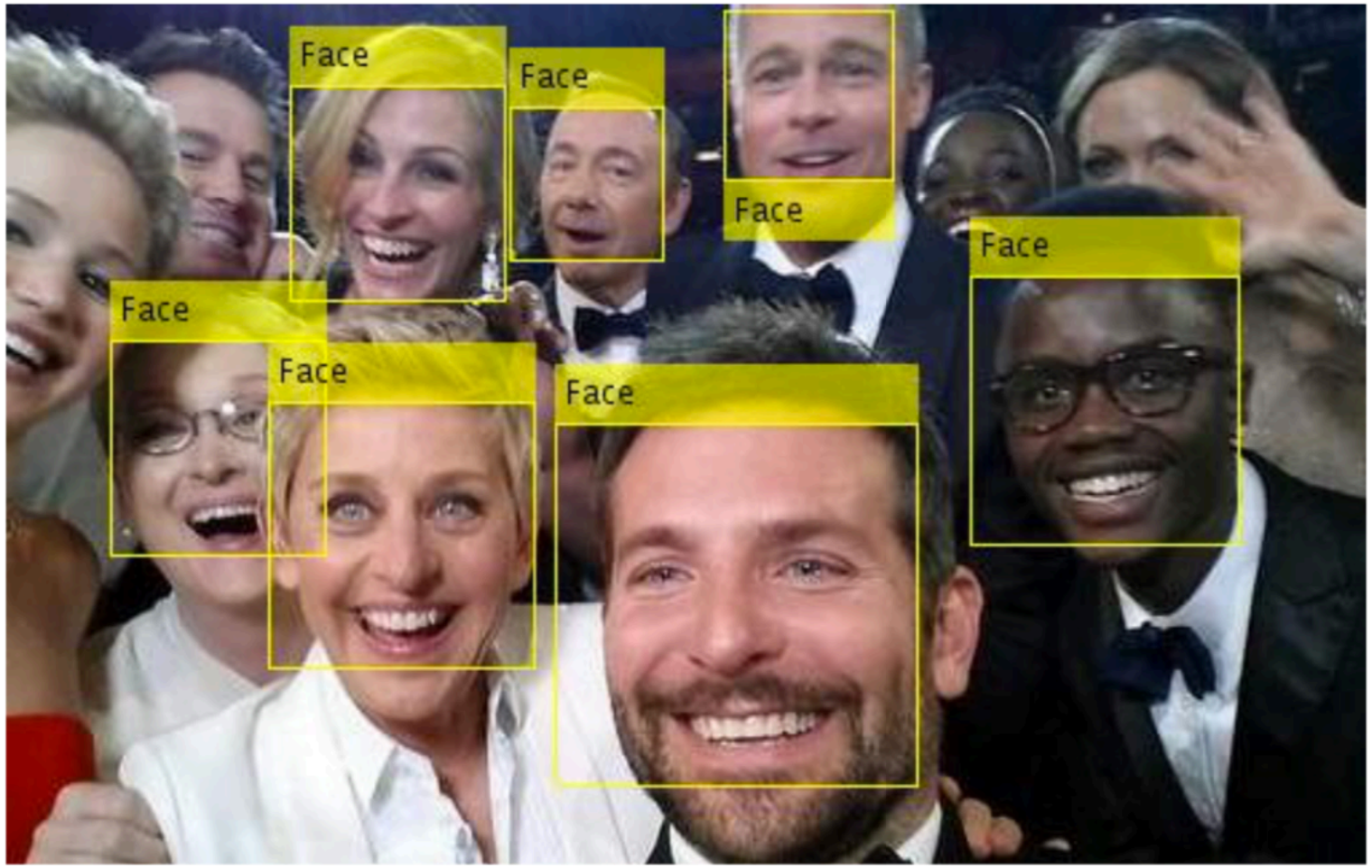
Boosting – like bagging – can reduce variance but – unlike bagging – can also eliminate the effect of high bias of weak learners.

Boosting vs Bagging

Bagging	Boosting
Learns base models in parallel	Learns base models sequentially
Uses bootstrapped datasets	Uses reweighted datasets
Reduces variance but not bias (e.g. it requires deep trees)	Reduces both variance and bias (e.g. it works well with shallow trees)

Source: Lindsten, Uppsala University

Application: The Viola-Jones Face Detector



Application: The Viola-Jones Face Detector

The Viola-Jones Face Detector

This detector – based on Adaboost – revolutionized computer face detection in 2001.

It uses extremely simple and fast base models:

(1) It places a mask of type A, B, C or D on top of the image

(2) It computes a value corresponding to the difference of intensity between sum of pixels in black area and sum of pixels in the white area

(3) It then classifies as face or no face

The base models are combined using Adaboost principles leading to a fast, accurate face detector

Masks