

Lecture 7 – Boosting



UPPSALA
UNIVERSITET

Thomas Schön

Division of Systems and Control
Department of Information Technology
Uppsala University.

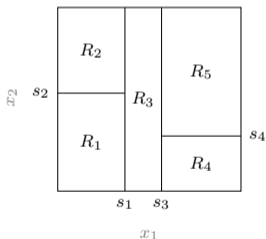
Email: thomas.schon@it.uu.se

Summary of Lecture 6 (I/III)

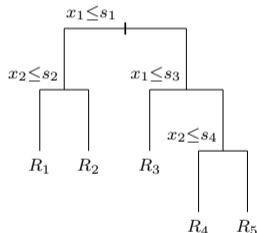
CART: Classification And Regression Trees

Partition the input space using **recursive binary splitting**

Partitioning of input space



Tree representation



- **Classification:** Majority vote within the region.
- **Regression:** Mean of training data within the region.

Summary of Lecture 6 (II/III)

Deep tree models tend to have **low bias** but **high variance**.

Bagging (=bootstrap aggregating) is a general **ensemble method** that can be used to reduce model variance (well suited for trees):

1. For $b = 1$ to B (*can run in parallel*)
 - (a) Draw a bootstrap data set $\tilde{\mathcal{T}}^b$ from \mathcal{T} .
 - (b) Train a model $\tilde{y}^b(\mathbf{x})$ using the bootstrapped data $\tilde{\mathcal{T}}^b$.
2. Aggregate the B models by outputting,
 - **Regression:** $\hat{y}_{\text{bag}}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^B \tilde{y}^b(\mathbf{x})$.
 - **Classification:** $\hat{y}_{\text{bag}}(\mathbf{x}) = \text{MajorityVote}\{\tilde{y}^b(\mathbf{x})\}_{b=1}^B$.

Summary of Lecture 6 (III/III)

Random forests = bagged trees, but with further variance reduction achieved by *decorrelating* the B ensemble members.

Specifically:

For each split in each tree only a *random subset* of $q \leq p$ inputs are considered as splitting variables.

Contents – Lecture 7

1. Boosting – *the general idea*
2. AdaBoost – *the first practical boosting method*
3. Robust loss functions
4. Gradient boosting

Boosting

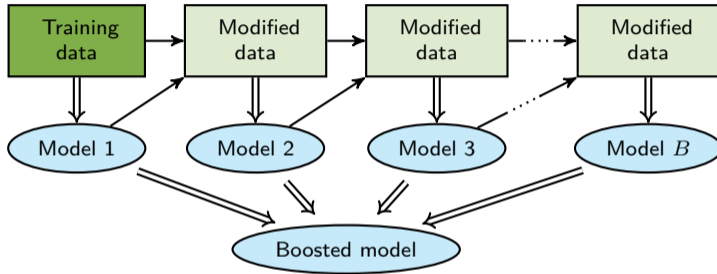
Even a simple (classification or regression) model can typically capture some aspects of the input-output relationship.

Can we then learn an **ensemble** of “weak models”, each describing some part of this relationship, and combine these into one “strong model”?

Boosting:

- **Sequentially** learns an ensemble of **weak models**.
- Combine these into one **strong model**.
- General strategy – can in principle be used to improve any supervised learning algorithm.
- One of the most successful machine learning ideas!

Boosting



The models are built **sequentially** such that each models tries to **correct the mistakes** made by the previous one!

Binary classification

We will restrict our attention to binary classification.

- Class labels are -1 and 1 , i.e. $y \in \{-1, 1\}$.
 - We have access to some (weak) base classifier, e.g. a classification tree.
-

Note. Using labels -1 and 1 is mathematically convenient as it allows us to express a majority vote between B classifiers $\hat{y}^1(\mathbf{x}), \dots, \hat{y}^B(\mathbf{x})$ as

$$\text{sign} \left(\sum_{b=1}^B \hat{y}^b(\mathbf{x}) \right) = \begin{cases} +1 & \text{if more plus-votes than minus-votes,} \\ -1 & \text{if more minus-votes than plus-votes.} \end{cases}$$

Boosting procedure (for classification)

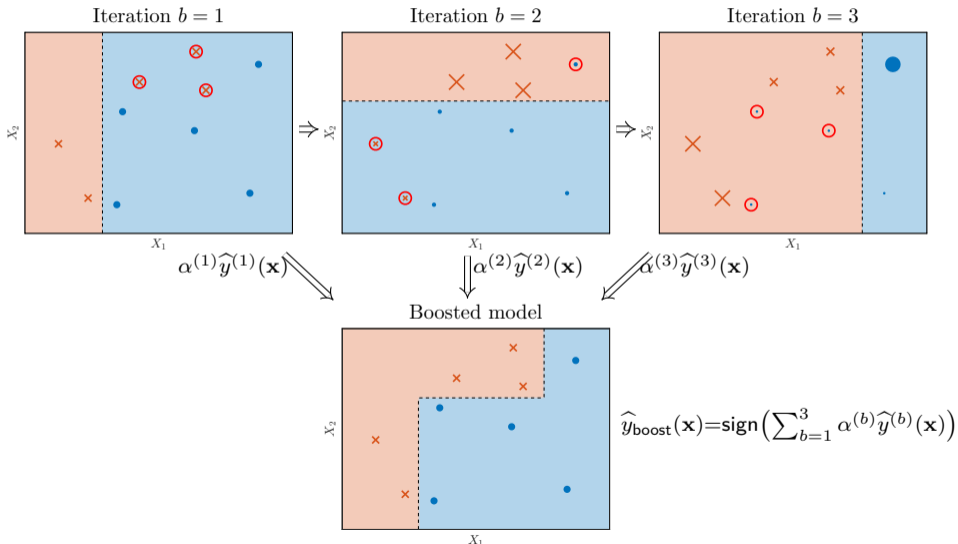
Boosting procedure:

1. Assign weights $w_i^1 = 1/n$ to all data points.
2. For $b = 1$ to B
 - (a) Train a weak classifier $\hat{y}^{(b)}(\mathbf{x})$ on the **weighted training data** $\{(\mathbf{x}_i, y_i, w_i^b)\}_{i=1}^n$.
 - (b) *Update the weights* $\{w_i^{b+1}\}_{i=1}^n$ from $\{w_i^b\}_{i=1}^n$:
 - i. Increase weights for all points misclassified by $\hat{y}^{(b)}(\mathbf{x})$.
 - ii. Decrease weights for all points correctly classified by $\hat{y}^{(b)}(\mathbf{x})$.

The predictions of the B classifiers, $\hat{y}^{(1)}(\mathbf{x}), \dots, \hat{y}^{(B)}(\mathbf{x})$, are combined using a **weighted** majority vote:

$$\hat{y}_{\text{boost}}^B(\mathbf{x}) = \text{sign} \left(\sum_{b=1}^B \alpha^{(b)} \hat{y}^{(b)}(\mathbf{x}) \right).$$

ex) Boosting illustration



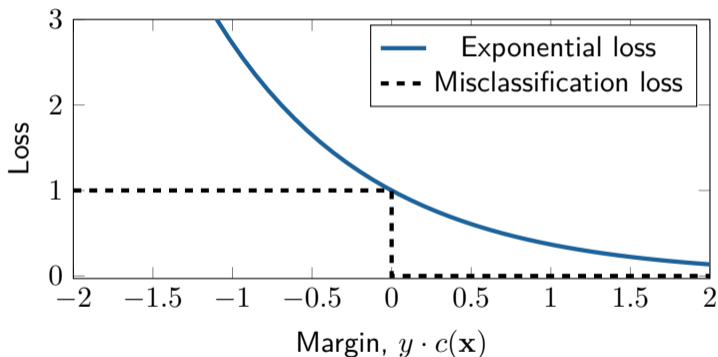
The technical details...

Q1: How do we reweight the data?

Q2: How are the coefficients $\alpha^{(1)}, \dots, \alpha^{(B)}$ computed?

Exponential loss

Loss functions for binary classifier $\hat{y}(\mathbf{x}) = \text{sign}(c(\mathbf{x}))$.



Exponential loss function $L(y, c(\mathbf{x})) = \exp(-y \cdot c(\mathbf{x}))$ plotted vs. **margin** $y \cdot c(\mathbf{x})$. The misclassification loss $\mathbb{I}\{y \neq \hat{y}(\mathbf{x})\} = \mathbb{I}\{y \cdot c(\mathbf{x}) < 0\}$ is plotted as comparison.

AdaBoost pseudo-code

1. Assign weights $w_i^1 = 1/n$ to all data points.
2. For $b = 1$ to B
 - (a) Train a weak classifier $\hat{y}^{(b)}(\mathbf{x})$ on the **weighted training data** $\{(\mathbf{x}_i, y_i, w_i^b)\}_{i=1}^n$.
 - (b) *Update the weights* $\{w_i^{b+1}\}_{i=1}^n$ from $\{w_i^b\}_{i=1}^n$:
 - i. Increase weights for all points misclassified by $\hat{y}^{(b)}(\mathbf{x})$.
 - ii. Decrease weights for all points correctly classified by $\hat{y}^{(b)}(\mathbf{x})$.
 - iii. Compute weighted classification error Compute $E_{\text{train}}^b = \sum_{i=1}^n w_i^b \mathbb{I}\{y_i \neq \hat{y}^{(b)}(\mathbf{x}_i)\}$
 - iv. Compute classifier "confidence" Compute $\alpha^b = 0.5 \log((1 - E_{\text{train}}^b)/E_{\text{train}}^b)$.
 - v. Compute new weights Compute $w_i^{b+1} = w_i^b \exp(-\alpha^{(b)} y_i \hat{y}^{(b)}(\mathbf{x}_i))$, $i = 1, \dots, n$
 - vi. *Normalize.* Set $w_i^{b+1} \leftarrow w_i^{b+1} / \sum_{j=1}^n w_j^{b+1}$, for $i = 1, \dots, n$.
3. Output $\hat{y}_{\text{boost}}^{(B)}(\mathbf{x}) = \text{sign}\left(\sum_{b=1}^B \alpha^{(b)} \hat{y}^{(b)}(\mathbf{x})\right)$.



2003 Gödel Prize

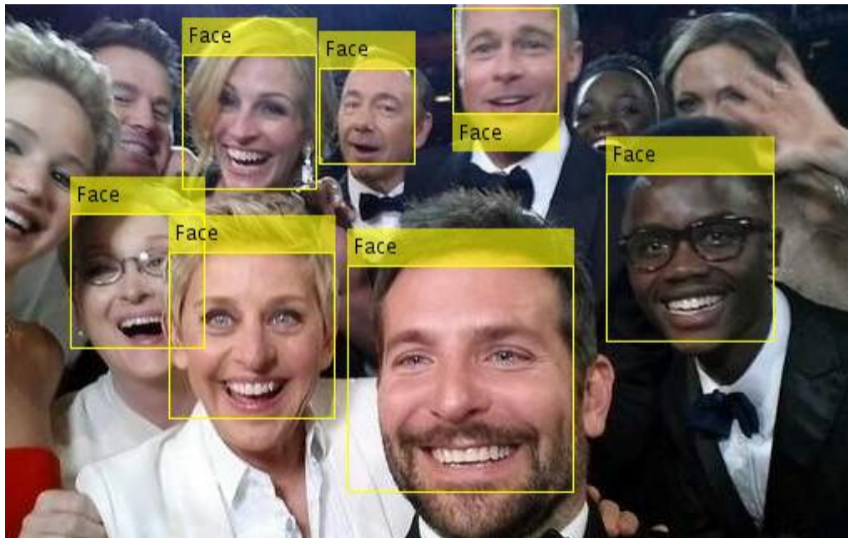
Y. Freund and R. E. Schapire. **Experiments with a New Boosting Algorithm.** *Proceedings of the 13th International Conference on Machine Learning (ICML)*. Bari, Italy, 1996.

Boosting vs. bagging

Bagging	Boosting
Learns base models in parallel	Learns base models sequentially
Uses bootstrapped datasets	Uses reweighted datasets
Does not overfit as B becomes large	Can overfit as B becomes large
Reduces variance but not bias (requires deep trees as base models)	Also reduces bias! (works well with shallow trees)

N.B. Boosting does *not* require each base model to have low bias. Thus, a shallow classification tree (say, 4-8 terminal nodes) or even a tree with a single split (2 terminal nodes, a “stump”) is often sufficient.

ex) The Viola-Jones face detector



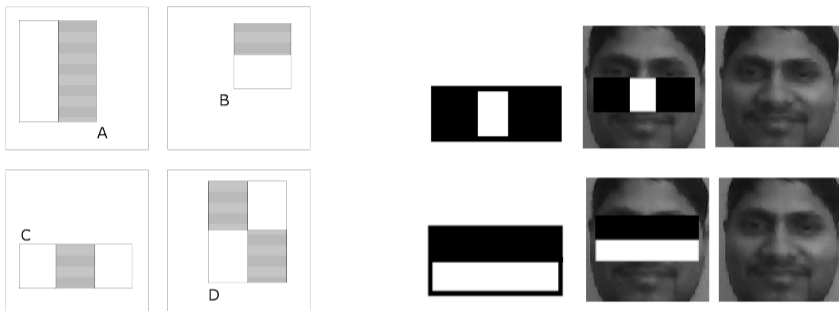
ex) The Viola-Jones face detector

The Viola-Jones face detector:

- Revolutionized computer face detection in 2001.
- First real-time system — soon built into standard point-and-shoot cameras.
- Based on AdaBoost!

P. Viola and M. Jones. **Rapid object detection using a boosted cascade of simple features**. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Kauai, Hawaii, USA, 2001.

ex) The Viola-Jones face detector



Uses *extremely simple (and fast!)* base models:

1. Place a “mask” of type *A*, *B*, *C* or *D* on top of the image
2. Value = \sum (pixels in black area) – \sum (pixels in white area)
3. Classify as face or noface.

Base models combined using **AdaBoost**, resulting in a fast and accurate face detector.

Robust loss functions

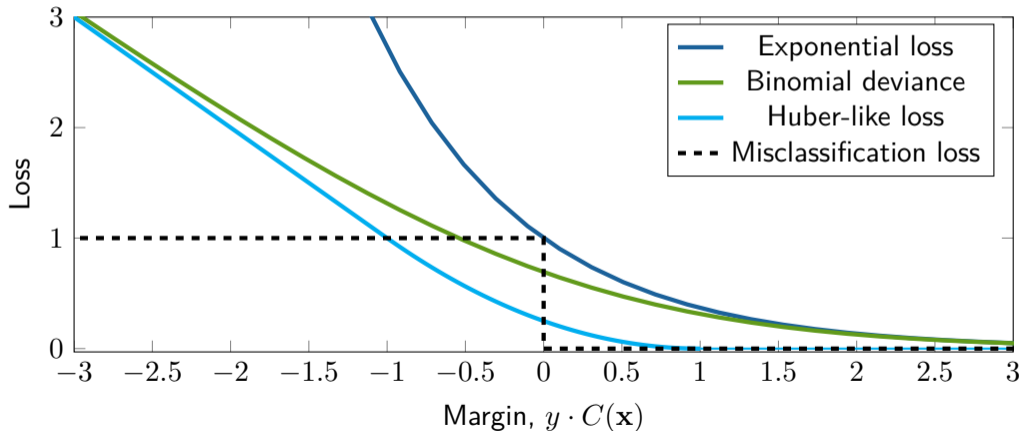
Why exponential loss?

- *Main reason* – computational simplicity (cf. squared loss in linear regression)

However, models using exponential loss is sensitive to “**outliers**”, e.g. mislabeled or noisy data.

Robust loss functions

Instead of exponential loss we can use a **robust loss function** with a more gentle slope for negative margins.



Gradient boosting

No free lunch! The optimization problem

$$(\alpha^{(b)}, \hat{y}^{(b)}) = \arg \min_{(\alpha, \hat{y})} \sum_{i=1}^n L(y_i, c^{(b-1)}(\mathbf{x}_i) + \alpha \hat{y}(\mathbf{x}_i))$$

lacks a closed form solution for a general loss function $L(y, c(\mathbf{x}))$.

Gradient boosting methods address this by using techniques from numerical optimization.

Gradient descent

Consider the generic optimization problem $\min_{\theta} J(\theta)$. A numerical method for solving this problem is **gradient descent**.

Initialize $\theta^{(0)}$ and iterate:

$$\theta^{(b)} = \theta^{(b-1)} - \gamma^{(b)} \nabla_{\theta} J(\theta^{(b-1)}).$$

where α^b is the **step size** at the b th iteration.

Can be compared with **boosting** – sequentially construct an ensemble of models as:

$$c^{(b)}(\mathbf{x}) = c^{(b-1)}(\mathbf{x}) + \alpha^{(b)} \hat{y}^{(b)}(\mathbf{x})$$

for some base model $\hat{y}^{(b)}(\mathbf{x})$ and “step size” $\alpha^{(b)}$.

Gradient boosting

Gradient boosting mimics the gradient descent algorithm by fitting the b th base model to the **negative gradient** of the loss function,

$$g_i^{(b)} = - \left[\frac{\partial L(y_i, c)}{\partial c} \right]_{c=c^{(b-1)}(\mathbf{x}_i)}, \quad i = 1, \dots, n.$$

That is, $\hat{y}^{(b)}(\mathbf{x})$ is learned using $\{(\mathbf{x}_i, g_i^{(b)})\}_{i=1}^n$ as training data.

Gradient boosting (with some additional bells and whistles) is used by the very popular libraries **XGBoost** and **LightGBM**,

`xgboost.readthedocs.io/`
`lightgbm.readthedocs.io/`

Classification methods (covered so far)

So far in the course we have discussed the following methods for classification. . .

Parametric classifiers

1. Logistic regression – **linear**
2. Linear discriminant analysis — **linear**
3. Quadratic discriminant analysis — **nonlinear**

Nonparametric classifiers

4. k -nearest neighbors
5. Classification trees

Ensemble-based methods

6. Bagging (bagged trees / random forests)
7. Boosting (AdaBoost)

A few concepts to summarize lecture 7

Boosting: Sequential ensemble method, where each consecutive model tries to correct the mistakes of the previous one.

AdaBoost: The first successful boosting algorithm. Designed for binary classification.

Exponential loss: The classification loss function used by AdaBoost.

Gradient boosting: A boosting procedure that can be used with any differentiable loss function.