Lecture 7 – Boosting

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

Email: fredrik.lindsten@it.uu.se
**Summary of Lecture 6 (I/III)**

**CART:** Classification And Regression Trees

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

- **Classification:** Majority vote within the region.
- **Regression:** Mean of training data within the region.
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 $\mathcal{T}^{*b}$ of size $n$ from $\mathcal{T}$.
   (b) Train a model ($\hat{f}^{*b}(X)$ for regression, $\hat{G}^{*b}(X)$ for classification) using the bootstrapped data $\mathcal{T}^{*b}$.

2. Aggregate the $B$ models by outputting,
   - **Regression**: $\hat{f}_{bag}(X) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(X)$.
   - **Classification**: $\hat{G}_{bag}(X) = \text{MajorityVote}\{\hat{G}^{*b}(X)\}_{b=1}^{B}$.
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!
The models are built **sequentially** such that each model tries to correct the mistakes made by the previous one!
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 $\hat{G}^1(X), \ldots, \hat{G}^B(X)$ as

$$\text{sign} \left( \sum_{b=1}^{B} \hat{G}^b(X) \right) = \begin{cases} +1 & \text{if more plus-votes than minus-votes}, \\ -1 & \text{if more minus-votes than plus-votes}. \end{cases}$$
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 $\hat{G}_b^b(X)$ on the weighted training data $\{(x_i, y_i, w_i^b)\}_{i=1}^n$.
   (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 $\hat{G}_b^b(X)$.
      ii. Decrease weights for all points correctly classified by $\hat{G}_b^b(X)$.

The predictions of the $B$ classifiers, $\hat{G}^1(X), \ldots, \hat{G}^B(X)$, are combined using a weighted majority vote:

$$\hat{G}_{\text{boost}}^B(X) = \text{sign} \left( \sum_{b=1}^B \alpha^b \hat{G}^b(X) \right).$$
ex) Boosting illustration

\[
\hat{G}_{\text{boost}}(x) = \text{sign} \left( \sum_{b=1}^{3} \alpha^b \hat{G}^b(x) \right)
\]
Q1: How do we reweight the data?

Q2: How are the coefficients $\alpha^1, \ldots, \alpha^B$ computed?
Exponential loss

Loss functions for binary classifier $\hat{G}(X) = \text{sign}(C(X))$.

Exponential loss function $L(Y, C(X)) = \exp(-YC(X))$ plotted vs. the margin $YC(X)$. The misclassification loss $I(Y \neq \hat{G}(X)) = I(YC(X) < 0)$ is plotted as comparison.
AdaBoost pseudo-code

**AdaBoost:**

1. Assign weights $w^1_i = 1/n$ to all data points.
2. For $b = 1$ to $B$
   
   (a) Train a weak classifier $\hat{G}^b(X)$ on the weighted training data
   
   $\{(x_i, y_i, w^b_i)\}_{i=1}^n$.

   (b) **Update the weights** $\{w^{b+1}_i\}_{i=1}^n$ from $\{w^b_i\}_{i=1}^n$:
      
      i. Compute $\text{err}^b = \sum_{i=1}^n w^b_i I(y_i \neq \hat{G}^b(x_i))$
      
      ii. Compute $\alpha^b = 0.5 \log((1 - \text{err}^b)/\text{err}^b)$.

      iii. Compute $w^{b+1}_i = w^b_i \exp(-\alpha^b y_i \hat{G}^b(x_i))$, $i = 1, \ldots, n$

      iv. **Normalize.** Set $w^{b+1}_i \leftarrow w^{b+1}_i / \sum_{j=1}^n w^{b+1}_j$, for $i = 1, \ldots, n$.

3. Output $\hat{G}^B_{\text{boost}}(X) = \text{sign} \left( \sum_{b=1}^B \alpha^b \hat{G}^b(X) \right)$.

---

*Proceedings of the 13th International Conference on Machine Learning (ICML)*
Bari, Italy, 1996.

---

fredrik.lindsten@it.uu.se

---

2003 Gödel Prize
## Boosting vs. bagging

<table>
<thead>
<tr>
<th>Bagging</th>
<th>Boosting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learns base models in parallel</td>
<td>Learns base models sequentially</td>
</tr>
<tr>
<td>Uses bootstrapped datasets</td>
<td>Uses reweighted datasets</td>
</tr>
<tr>
<td>Reduces variance but not bias</td>
<td>Also reduces bias!</td>
</tr>
<tr>
<td>(requires deep trees as base models)</td>
<td>(works well with shallow trees)</td>
</tr>
</tbody>
</table>

**N.B.** Boosting does *not* require each base classifier 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.
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!

ex) The Viola-Jones face detector

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

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.

The base classifiers are combined using **AdaBoost**, resulting in a fast and accurate face detector.

fredrik.lindsten@it.uu.se
Robust loss functions

Why exponential loss?

- **Main reason** – computational simplicity (cf. squared loss in linear regression)
- In the large data limit, minimizing the exponential loss gives half the log-odds

\[
C(X) = \frac{1}{2} \log \left[ \frac{\Pr(Y = 1 \mid X)}{\Pr(Y = -1 \mid X)} \right].
\]

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

$$(\alpha^b, \hat{G}^b) = \arg \min_{(\alpha, G)} \sum_{i=1}^{n} L(y_i, C^b(x_i))$$

lacks a closed form solution for a general loss function $L(Y, C(X))$.

**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:

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

where $\alpha^b$ is the step size at the $b$th iteration.

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

$$
C^b(X) = C^{b-1}(X) + \alpha^b \hat{f}^b(X)
$$

for some base model $\hat{f}^b(X)$ and “step size” $\alpha^b$. 
Gradient boosting mimics the gradient descent algorithm by fitting the $b$th base model to the negative gradient of the loss function,

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

That is, $\hat{f}^b(X)$ is learned using $\{(x_i, g_i^b)\}_{i=1}^n$ as training data.

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

https://github.com/dmlc/xgboost/tree/master/demo
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)
**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.