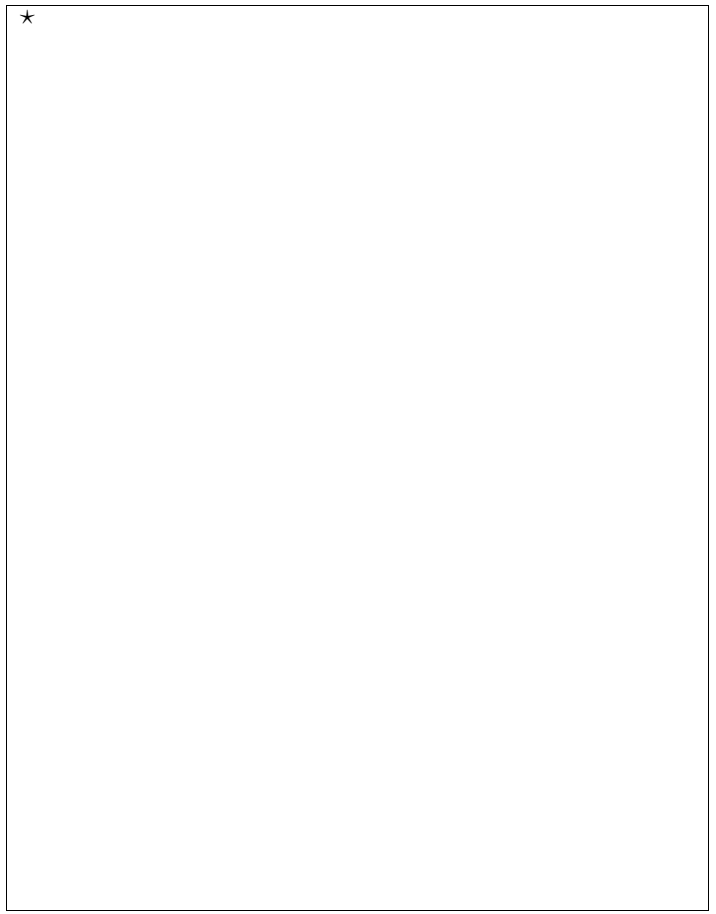
Lecture 11 - *Boosting and Trees*

OBJECTIVE: In this lecture, we learn to classify data using simple decision trees (stumps). We then use a sequential minmax algorithm, called boosting, to combine these simple trees into more complex trees. That is, the idea of boosting is to combine simple classifiers to produce more sophisticated ones.

Textbook: Pages 266–279 and 299 to 345.

Decision Stumps

A decision stump partitions the feature space into two halves. It assigns the label -1 to points in one half and +1 to points in the other half. The algorithm works as follows.



At each splitting point, the decision stump estimates the probability of the points in region 1 being equal to 1 and -1 as follows:

$$p(y = 1 \text{ in region } R_1) = \frac{1}{N_1} \sum_{x_i \in R_1} \mathbb{I}(y_i = 1)$$

$$p(y = -1 \text{ in region } R_1) = \frac{1}{N_1} \sum_{x_i \in R_1} \mathbb{I}(y_i = -1)$$

where N_1 is the number of points in region 1. We then assign to region 1 the label with the highest probability.

Lets assume that after computing the above probabilities, the label of region 1 is +1 and the label of region 2 is -1. The total error is then computed as follows:

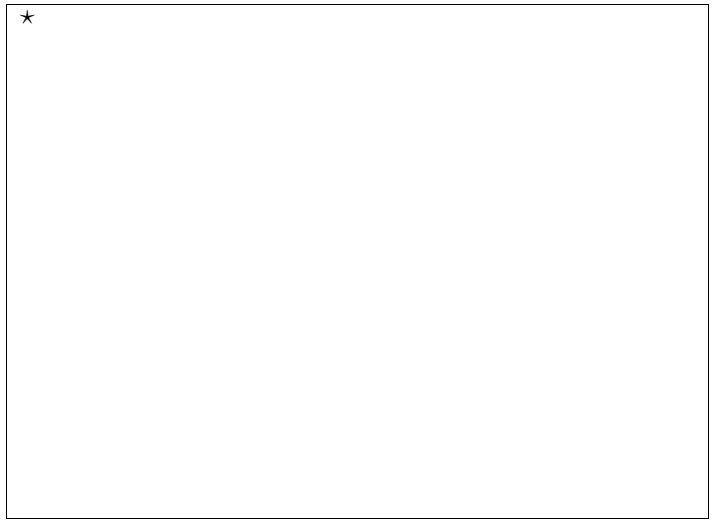
$$error = \frac{1}{N_{R_1}} \sum_{x_i \in R_1} \mathbb{I}(y_i \neq +1) + \frac{1}{N_{R_2}} \sum_{x_i \in R_2} \mathbb{I}(y_i \neq -1)$$

Note that the error is a function of where we place the stump (decision boundary) and what feature (or dimension) we are considering. Therefore, we need to compute the errors for

each dimension and for each position of the stump. We, finally, pick the stump that results in the lowest error.

Weighted Decision Stumps

Not all points are born equal. Some are more important. Let's look at an example:



Points near the boundary are more critical to classification. We saw this before in the context of SVMs. It is possible to

account for this with decision stumps by adding a weight w_i to each feature x_i. We then seek to minimise the following error:

$$e_w = \frac{1}{\sum_{x_i \in R_1} w_i} \sum_{x_i \in R_1} w_i \mathbb{I}(y_i \neq +1) + \frac{1}{\sum_{x_i \in R_2} w_i} \sum_{x_i \in R_2} w_i \mathbb{I}(y_i \neq -1)$$

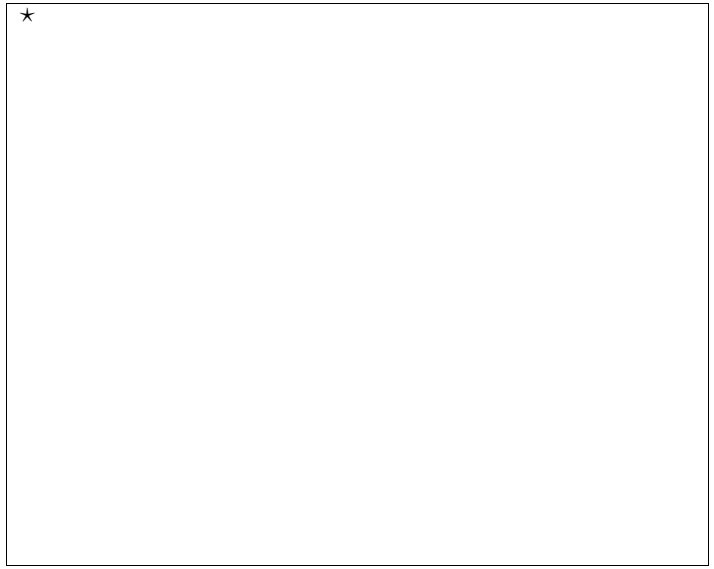
Boosting

Boosting is a sequential minimax technique for combining M simple (weak) classifiers into a powerful (strong) one. If the m-th weak classifier is denoted by $G_m(x)$, then the strong classifier is:

$$G(x) = sign\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$$

where α_m is a learned coefficient that is larger for better (accurate) weak classifiers.

Pictorially, the algorithm proceeds as follows:



Boosting starts by assigning uniform weights $w_i = 1/N$ to each data point $x_1, x_2, \ldots, x_N$. At each successive iteration m, the weights are changed so that points that were misclassified get a higher weight for the next iteration. That is, boosting focus increasingly on the points that are harder to classify.

The pseudo-code for AdaBoost (a variant of boosting in-

troduced by Rob Schapire) follows:

1. `Initialize weights:` $w_i = 1/N$ `for` $i = 1, \ldots, N$.

2. `For m=1 to M:`

   (a) `Fit a classifier` $G_m(x)$ `with weights` w_i.

   (b) `Compute the error of the weak classifier:`

$$error_m = \frac{\sum_{i=1}^{N} w_i \mathbb{I}\left(y_i \neq G_m(x_i)\right)}{\sum_{1=1}^{N} w_i}$$

   (c) `Compute classifier coefficients:`

$$\alpha_m = \log\left(\frac{1 - error_m}{error_m}\right)$$

   (d) `Update the weights:`

$$w_i \longleftarrow w_i \exp\left[\alpha_m \mathbb{I}\left(y_i \neq G_m(x_i)\right)\right]$$

3. `Output` $G(x) = sign\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$