Gradient Boosting - ADABoost

umberto fontana

December 2022

1 Introduction

AdaBoost ("adaptive boosting") is an ensemble method for classification (or regression) that:

- trains multiple learners on weighted sample points;
- uses different weights for each learner;
- increases weights of misclassified training points;
- gives bigger votes to more accurate learners.

Consider X input data matrix with shape nxd, $y \in \{-1, +1\}$.

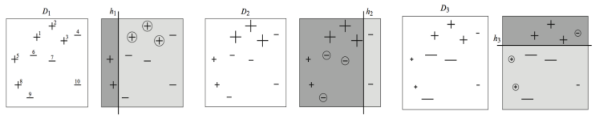


Figure 1:

The main idea is to fit an additive model (ensemble) $\sum_t \rho_t h_t(x)$ in a forward stage-wise manner. In each stage, we introduce a weak learner to compensate the shortcomings of existing weak learner.

2 Comprehensive example

You're given $(x_1, y_1), ..., (x_n, y_n)$ and the task is to fit a model $F(x)$ to minimize the square loss.

Suppose your friend wants to help you and gives you a model F. You check his model and find that the model is good but not perfect (for example, $F1(x_1) = 0.8 but y_1 = 0.9$. How can you improve this model?

You have some limitations. You are not allowed to remove anything from F or change any parameter in F. Furthermore, you can add an additional model (regression tree) h to F, so the new prediction will be $F(x) + h(x)$.

You wish, for each i:
F(x$_i$) $+ h(x_i) = y_i$
or, equivalently:
h(x$_i$) $= y_i - F(x_i)$
Can any regression tree h achieve this goal perfectly? Some regression tree might be able to do this approximately by just fit a regression tree h to data $(x_i, y_i - F(x_i))$. This is a better model.
$y_i - F(x_i)$ are called **residuals**. These are the parts that existing model F cannot do well. If the new model $F + h$ is still not satisfactory, we can add another regression tree...

3 Gradient Boosting for Regression

We have a loss function

$$L(y, F(x)) = (y - F(x))^2/2$$

and we want to minimize $J = \sum_i L(y_i, F(x_i))$ by adjusting $F(x_i)$ for all i.
Notice that $F(x_i)$ are just some numbers and we can treat them as parameters and take the derivatives.

$$\frac{\partial J}{\partial F(x_i)} = \frac{\partial \sum_i L(y_i, F(x_i))}{\partial F(x_i)} = \frac{\partial L(y_i, F(x_i)}{\partial F(x_i)} = F(x_i) - y_i.$$

Recall that for the gradient descent we move in the negative direction. For the correction:

$$F(x_i) := F(x_i) + h(x_i) F(x_i) := F(x_i) + y_i - F(x_i) F(x_i) := F(x_i) - 1\frac{\partial J}{\partial F(x_i)}$$

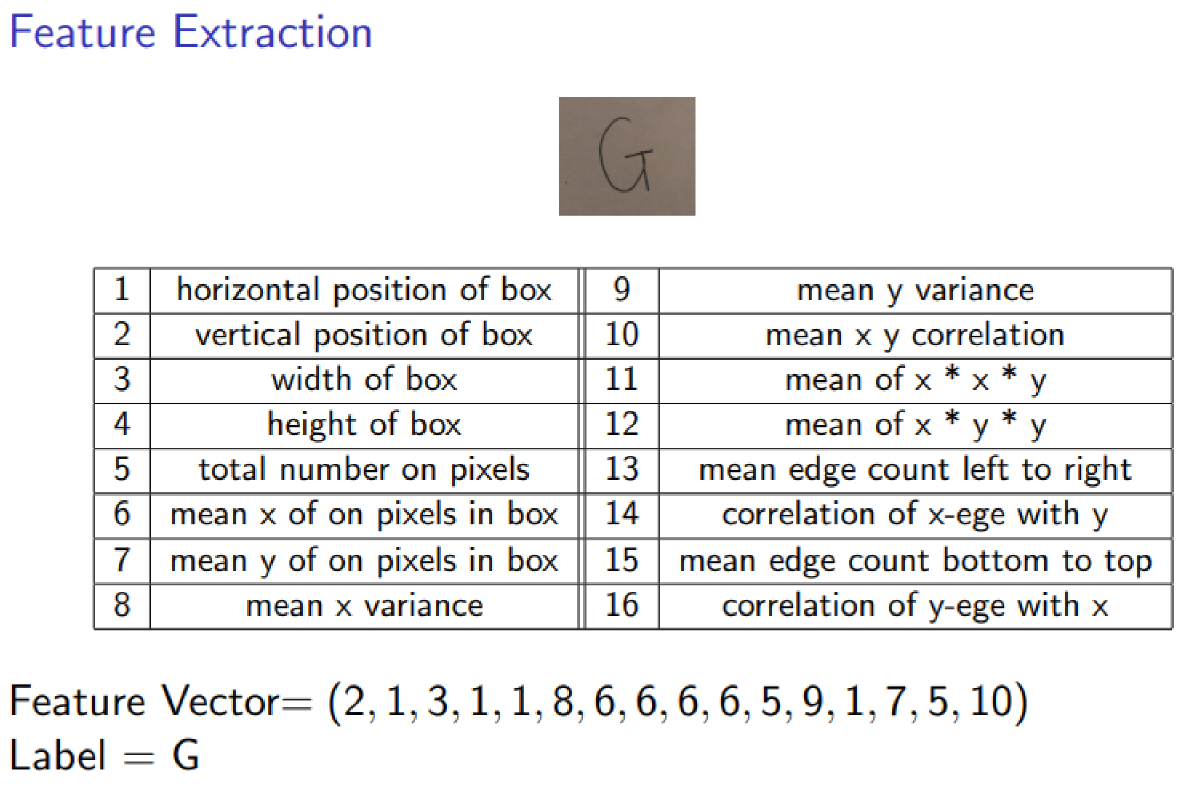
The forward expression is the classical updating of the parameters. So we are actually updating our model using gradient descent.
So in this case, the main step will be:

1. Start with the initial model F
2. Calculate the negative gradient $-g(x_i) = y_i - F(x_i)$
3. Fit a regression tree h to negative gradients $-g(x_i)$
4. $F := F + \rho h$ where $\rho = 1$.

It is important to notice that the concept of residual coincides with the residual's one only for the square loss. It is generally different for, for example, the Humber Loss or the Absolute Loss. In any case, the regression tree h will fit on the negative gradient $-g(x_i)$.

4 Gradient Boosting for Classification

Let's consider the problem of recognize a capital letter (26 classes).



Feature Extraction

1	horizontal position of box	9	mean y variance
2	vertical position of box	10	mean x y correlation
3	width of box	11	mean of x * x * y
4	height of box	12	mean of x * y * y
5	total number on pixels	13	mean edge count left to right
6	mean x of on pixels in box	14	correlation of x-ege with y
7	mean y of on pixels in box	15	mean edge count bottom to top
8	mean x variance	16	correlation of y-ege with x

Feature Vector= (2, 1, 3, 1, 1, 8, 6, 6, 6, 6, 5, 9, 1, 7, 5, 10)
Label = G

Figure 2:

Our model will have 26 score functions (our models): $F_A, F_B, ..., F_Z$. $F_A(x)$ assigns a score for class A.

Scores are used to calculate probabilities:

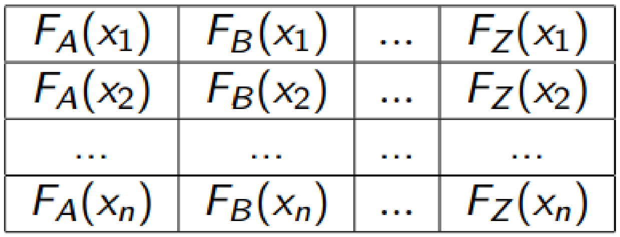
$$P_I(x) = \frac{e^{F_I(x)}}{\sum_{c=A}^{Z} e^{F_c(x)}}$$

The predicted label will be the class with the highest probability.

So the steps will be:

1. turn y_i into a true probability distribution $Y_c(x_i)$, for example, if $y_5 = G$, $Y_A(x_5) = 0, Y_B(x_5) = 0, ...Y_G(x_5) = 1, ..., Y_Z(x_5) = 0$.

2. calculate the predicted probability distribution $P_c(x_i)$ based on the current model $F_A, ..., F_Z$.



$F_A(x_1)$	$F_B(x_1)$	...	$F_Z(x_1)$
$F_A(x_2)$	$F_B(x_2)$	...	$F_Z(x_2)$
...	...	...	...
$F_A(x_n)$	$F_B(x_n)$	...	$F_Z(x_n)$

Figure 3:

3. calculate the difference between the true probability distribution and the predicted probability distribution. (here we use KL-divergence).

The difference with respect to the regression, is that we have a matrix of parameters to optimize instead of a column of parameters to optimize.

And this will need multiple regression trees h to fit multiple negative gradients $-g(x_i)$.