Boosting 1

-Statistical Machine Learning-

Lecturer: Darren Homrighausen, PhD

Boosting overview

Recall: Bagging is a procedure for taking a low bias, high variance procedure and (potentially) reducing its risk via averaging

Boosting has a similar philosophy: take a poor classifier and improve it

However, boosting is useful for the opposite situation: a classifier that has high bias but low variance!

Boosting overview

A direct contrast:

- Bagging: aggregates over many independent bootstrap draws
- Boosting: finds the observations that are poorly classified, up weights these observations, and then trains a new classifier

Boosting for Regression

Boosting for regression

There are three main ingredients to boosting:

- A base learner $\hat{f}$

  (This $\hat{f}$ will commonly have some parameters determining its complexity. These are commonly set at very low complexity values)
- A learning rate λ
- The number of base learners B

  (This will act a bit like the number of iterations for random forest. However, the details are quite different)

Boosting regression trees

A classic example of a base learner is (regression) trees

Recall: Trees tend to have a low bias but high variance. This makes them well-suited for boosting

Before discussing boosting further, it is instructive to examine a basic implementation

(We will get to motivation and classification later)

Boosting regression trees

Set $\hat{f} \equiv 0$ and $R = Y \in R^n$

Fix the tree complexity M and learning rate λ

(Small values of M are used, such as $M \in \{1, \ldots, 8\}$, where M is the number of splits)

For $b = 1, \ldots, B$, do:

1. Fit $\hat{f}_b$ with $M + 1$ regions to $\tilde{\mathcal{D}} = \{(X_1, R_1), \ldots, (X_n, R_n)\}$
2. Update: $\hat{f} \leftarrow \hat{f} + \lambda \hat{f}_b$
3. Update: $R \leftarrow R - \hat{f}$

Output:

$$\hat{f} = \sum_{b=1}^{B} \lambda \hat{f}_b$$

This is an additive model

Boosting trees

In general

- A smaller λ means a larger required B
- Too large of λ means we take too long of steps, leading to poor solutions

  (Recall: gradient descent)

In practice,

- B is set via cross-validation or other risk estimate

  (Boosting is largely insensitive to overfitting by choosing B too large)
- λ is set at a small level, say $\lambda = 0.01$

As for the additive model part...

Curse of dimensionality and local averaging

From linear to nonlinear models

Goal: Develop a prediction function $\hat{f} : \mathbb{R}^p \rightarrow \mathbb{R}$ for predicting Y given an X

Commonly, $\hat{f}(X) = X^\top \beta$
(Constrained linear regression)

This greatly simplifies algorithms, while not sacrificing too much flexibility

However, sometimes directly modeling the nonlinearity is more natural

Prediction via local averaging

The fundamental quantities of interest we have been modeling are the Bayes' rules

$$\mathbb{E}[Y|X] \quad \text{or} \quad \arg\max_g \mathbb{P}(Y = g|X)$$

We know how to estimate expectations: if $Y_1, Y_2, \ldots, Y_n$ all have expectation μ, then

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} Y_i$$

is an intuitive estimator of μ

(and a reasonable prediction of a new Y)

Prediction via local averaging

Similarly, we can estimate $\mathbb{E}[Y|X]$ with $\mathcal{D}$:

$$\hat{f}(X) = \frac{1}{n_X} \sum_{i=1}^{n_X} Y_i \mathbf{1}(X_i = X)$$

where $n_X = \sum_{i=1}^{n} \mathbf{1}(X_i = X)$.

(In words: we are taking an average of all the observations Y_i such that $X_i = X$. This is all conditional expectation really is)

Prediction via local averaging

There is a problem: There generally aren't any X_i at X!

Suppose we relax the constraint $X_i = X$ a bit and include points that are close enough instead

Again, suppose we have data $(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)$

$$\hat{f}(X) = \frac{1}{n_X} \sum_{i=1}^{n_X} Y_i \mathbf{1}(||X_i - X|| \leq t)$$

where $n_X = \sum_{i=1}^{n} \mathbf{1}(||X_i - X|| \leq t)$.

Here, t quantifies the notion of closeness

(In fact, it is a tuning parameter)

Prediction via local averaging

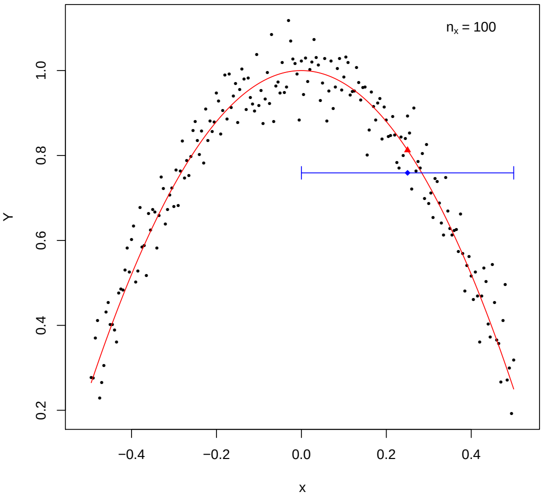


Figure: $t = 0.25$

Prediction via local averaging

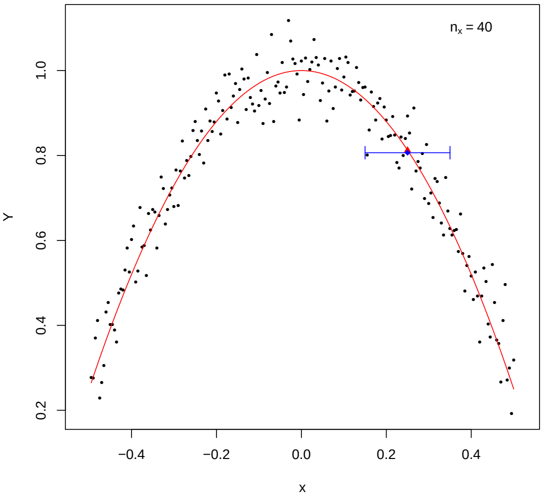


Figure: $t = 0.1$

Prediction via local averaging

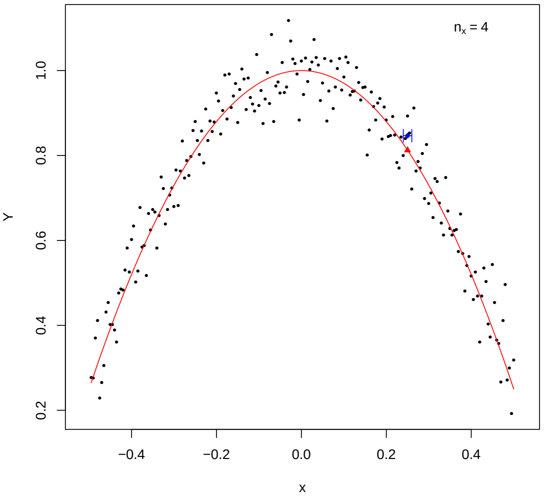


Figure: $t = 0.01$

Prediction via local averaging

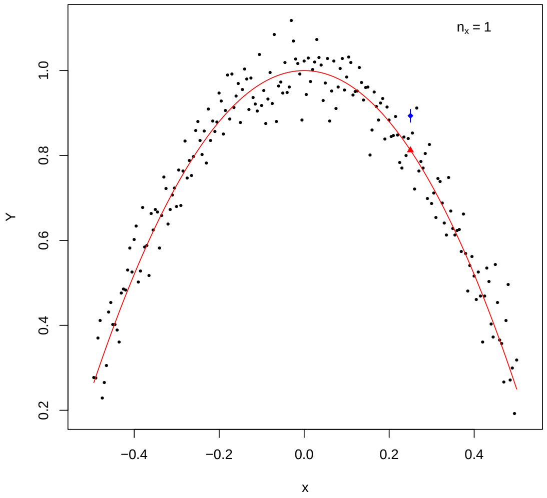


Figure: $t = 0.0001$

From linear to nonlinear models

Question: Why don't we always fit such a flexible model?

Answer: This works great if p is small

(and the specification of nearness is good)

However, as p gets large

- nothing is nearby
- all points are on the boundary

  (Hence, predictions are generally extrapolations)

These aspects make up (part) of the curse of dimensionality

Curse of dimensionality

Fix the dimension p

(Assume p is even to ignore unimportant digressions)

Let S be a hypersphere with radius r

Let C be a hypercube with side length $2r$

Then, the volume of S and C are, respectively

$$V_S = \frac{r^p \pi^{p/2}}{(p/2)!} \text{ and } V_C = (2r)^p$$

(Interesting observation: this means for $r < 1/2$ the volume of the hypercube goes to 0, but the diagonal length is always $\propto \sqrt{p}$. Hence, the hypercube gets quite 'spiky' and is actually horribly jagged. Regardless of radius, the hypersphere's volume goes to zero quickly.)

Curse of dimensionality

Hence, the ratio of the volumes of a circumscribed hypersphere by a hypercube is

$$\frac{V_C}{V_S} = \frac{(2r)^p \cdot (p/2)!}{r^p \pi^{p/2}} = \frac{2^p \cdot (p/2)!}{\pi^{p/2}} = \left(\frac{4}{\pi}\right)^d d!$$

where $d = p/2$

Observation: This ratio of volumes is increasing really fast. This means that all of the volume of a hypercube is near the corners. Also, this is independent of the radius.

Additive models

Additive models

We can find a combination of linear models and nonlinear models that provides flexibility while shielding us somewhat from the dimension problem

Write

$$f(X) = f_1(x_1) + \cdots + f_p(x_p) = \sum_{j=1}^{p} f_j(x_j)$$

Estimation of such a function is not much more complicated than a fully linear model (as all inputs enter separately)

The algorithmic approach is known as backfitting

Additive models (for regression)

Additive models are usually phrased using the population level expectation

(These get replaced with empirical versions)

The update is a Gauss-Seidel-type update

(The Gauss-Seidel method is an iterative scheme for solving linear, square systems)

This is for $j = 1, \ldots, p, 1, \ldots, p, 1 \ldots$:

$$f_j(x_j) \leftarrow \mathbb{E}\left[Y - \sum_{k \neq j} f_k(x_k) | x_j \right]$$

Under fairly general conditions, this converges to $\mathbb{E}[Y|X]$

Additive models (for regression)

Backfitting for additive models is roughly as follows:

Choose a univariate nonparametric smoother $\mathcal{S}$ and form all marginal fits $\hat{f}_j$

(Commonly a cubic smoothing spline)

Iterate over j until convergence:

1. Define the residuals $R_i = Y_i - \sum_{k \neq j} \hat{f}_k(X_i^k)$
2. Smooth the residuals $\hat{f}_j = \mathcal{S}(R)$
3. Center $\hat{f}_j \leftarrow \hat{f}_j - n^{-1} \sum_{i=1}^{n} \hat{f}_j(X_i^j)$

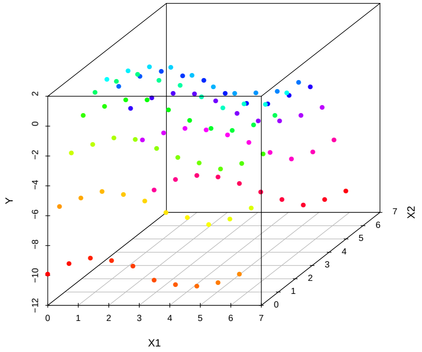
Report

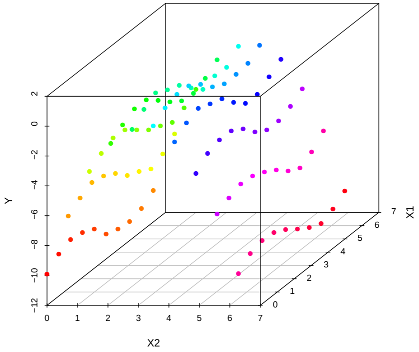
$$\hat{f}(X) = \overline{Y} + \hat{f}_1(x_1) + \cdots + \hat{f}_p(x_p)$$

Fitting additive models R

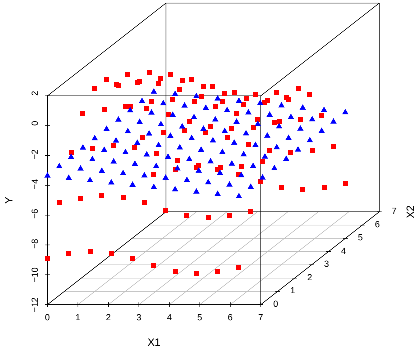
```
library(gam)
x    = seq(0,2*pi,length=10)
xx = expand.grid(x,x)
X1 = xx[,1]
X2 = xx[,2]

Y =  sin(xx[,1]) - (xx[,2] - pi)^2 + rnorm(nrow(xx),0,.1)
sim = data.frame(X1=X1,X2=X2,Y=Y)

out = gam(Y~s(X1,3)+s(X2,3),data=sim)
```

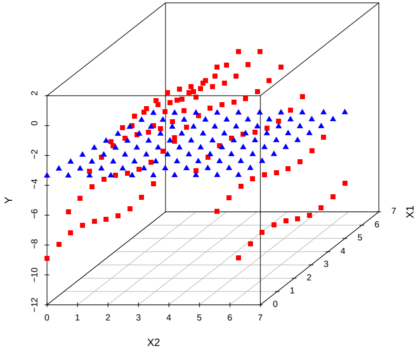
Additive models: Simulation





Additive models: Simulation results





(These are the fitted values only. Red squares: GAM, Blue triangles: multiple linear regression)

Detour: Plotting 3d in R

```
out = scatterplot3d(X1,X2,Y,pch=16,type='n')
xyz = out$xyz.convert(X1,X2,out.pred)
points(xyz,col='red',pch=15)
xyz = out$xyz.convert(X1,X2,out.pred.lm)
points(xyz,col='blue',pch=17)
```

Additive models (for regression)

More generally, we can consider each function in the sum to be a function of all input variables

Example: $f_b(X) = f_b(x_b)$

(That is, the function only depends on one component of X)

The resulting model would be

$$\sum_{b=1}^{B} f_b(X)$$

How can we fit this?

(Functional) Gradient descent

Let $\ell(f, Y)$ be a loss function and R be the risk

Example: $\ell(f, Y) = (f(X) - Y)^2$ and $R(f) = \mathbb{P}\ell(f, Y)$

Out goal is to minimize $R(f)$ over f.

Example: For squared error loss, the minimizer is $\mathbb{P}Y|X$

How about in general?

(Functional) Gradient descent

Form the gradient:

$$g = \frac{\partial R}{\partial f} = \mathbb{P}\frac{\partial \ell(f, Y)}{\partial f}$$

For $b = 1, \ldots, B$

$$f_b = f_{b-1} - \lambda g \Big|_{f = f_{b-1}}$$

It can be shown that by taking B large enough, $f_B \to \mathbb{P} Y | X$

(Functional) Gradient descent

The previously written algorithm isn't usable with data
(We need to estimate $\mathbb{P}$)

If we instead use

$$\hat{g}(X_i) = \frac{\partial \ell(f(X_i), Y_i)}{\partial f}$$

for $i = 1, \ldots, n$

This procedure both overfits and is only defined at the observed X_i

(Functional) Gradient descent

A way of preventing the overfitting is to restrict the subspace of functions we are looking at

Let $\mathcal{F}$ be a class of functions

After forming $\hat{g}$, we restrict it via projection to be in $\mathcal{F}$

(This grabs the element of $\mathcal{F}$ most parallel to $\hat{g}$)

(Functional) Gradient descent

A data-based algorithm is now: For $b = 1, \ldots, B$, do:

1. $R_i \leftarrow -\hat{g}(X_i)\Big|_{f=\hat{f}_{b-1}} = \frac{\partial \ell(f(X_i), Y_i)}{\partial f}\Big|_{f=\hat{f}_{b-1}}$
2. $\hat{f} \leftarrow \text{argmin}_{f \in \mathcal{F}} ||R - f||_2^2$
   (Projection step, allowing for $\hat{f}$ to be defined at new X)
3. Update: $\hat{f}_b \leftarrow \hat{f}_{b-1} + \lambda \hat{f}$

(Functional) Gradient descent

Let's look at step 1. more closely:

$$\frac{\partial \ell(f(X_i), Y_i)}{\partial f} = \frac{\partial (f(X_i) - Y_i)^2}{\partial f} = 2(f(X_i) - Y_i)$$

Observation: These are (twice) the residuals

(Hence, as in SVM, usually we use $(f(X) - Y)^2/2$)

(Functional) Gradient descent

Reminder: Back to boosting. Fix any b

1. Fit $\hat{f}_b$ with $M+1$ regions to $\tilde{\mathcal{D}} = \{(X_1, R_1), \ldots, (X_n, R_n)\}$
2. Update: $\hat{f} \leftarrow \hat{f} + \lambda \hat{f}_b$
3. Update: $R \leftarrow R - \hat{f}$

Compare: Functional gradient descent:

1. $R_i \leftarrow -\frac{\partial \ell(f(X_i), Y_i)}{\partial f}\bigg|_{f=\hat{f}_{b-1}} = 2(Y_i - f(X_i))$
2. $\hat{f} \leftarrow \text{argmin}_{f \in \mathcal{F}} ||R - f||_2^2$

   (Projection step, let $\mathcal{F}$ be class of trees with $M+1$ regions)
3. Update: $\hat{f}_b \leftarrow \hat{f}_{b-1} + \lambda \hat{f}$

(Functional) Gradient descent

Conclusion: These approaches are the same!

Boosting is an algorithmic way of fitting a general additive model using data

Now, we need to transfer this insight to classification..