MATH 343W Lec 23

Bagging: meta-algorithm

$g_{BAG} := \frac{g_1 + g_2 + \ldots + g_M}{M}$ for regression

$g_{BAG} :=$ Mode $[g_1, g_2, \ldots, g_M]$ for classification

The model g_i is called the "base-learner" which could be the result of any algorithm.

Which base learners do well? Those that have low bias and high variance and are as independent as possible ("strong learners")

Boosting: another meta-algorithm

$g_{boost} := g_1 + g_2 + \ldots + g_M$

where g_2 is fit on where $\vec{\hat{y}}_1 := g_1(X)$ doesn't fit,

g_3 " " " " " " " " $\vec{\hat{y}}_2 := g_1(X) + g_2(X)$ doesn't fit

$\vdots$

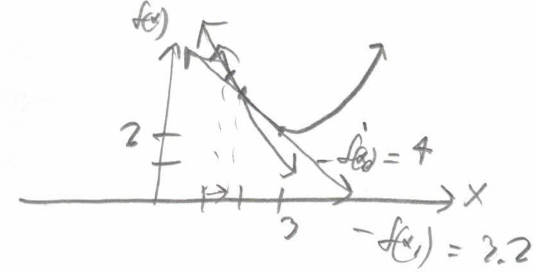
g_M " " " " " " " $\vec{\hat{y}}_{M-1} := g_1(X) + g_2(X) + \ldots + g_{M-1}(X)$ " " "

which is "iterative self-correction"

There are many ways to implement boosting. We will study "gradient boosting". What does "gradient" mean.

Go back to calculus...

Consider $f(x) = 2 + (x-3)^2$



We want to find

$x_* := \underset{x \in \mathbb{R}}{\arg\min} \{ f(x) \}$ or at least a local minimum

We first solve for derivative $f'(x) = 2(x-3)$

Pretend that you cannot solve for x if $f'(x) = 0$.

So... start at initial value $x_0 \overset{e.g.}{=} 1$

Here $-f'(x_0) = -2((1)-3) = 4$

iterate to get closer which is defined as a

If a "step" multiple of the negative derivative, we get closer to the local minimum. Let $\eta = 0.1$, the step size.

$x_1 = x_0 + \eta(-f'(x_0)) = 1 + 0.1 \cdot 4 = 1.4$, $-f'(x_1) = -2((1.4-3)) = 3.2$

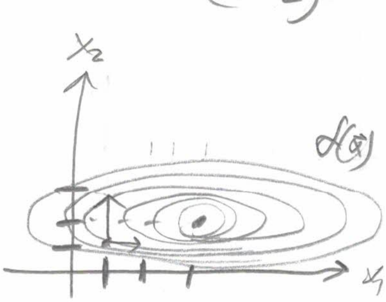
We iterate again

$\Rightarrow x_2 = x_1 - \eta f'(x_1) = 1.4 + 0.1 \cdot 3.2 = 1.72$ This is called "gradient descent"

In two dimensions e.g. $f: \mathbb{R}^2 \to \mathbb{R}$ $f(\vec{x}) = 2 + (x_1-3)^2 + (x_2-2)^4$

$\vec{x}_* := \arg\min\{f(\vec{x})\}$ will be a 2-dim vector

We first solve for $\vec{\nabla} f(\vec{x}) := \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2(x_1-3) \\ 4(x_2-2) \end{bmatrix}$



We start at $\vec{x}_0 \overset{e.g.}{=} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, $-\vec{\nabla} f(\vec{x}_0) = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$

Now we step $\vec{x}_1 = \vec{x}_0 - \eta \vec{\nabla} f(\vec{x}_0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} + 0.1 \begin{pmatrix} 3 \\ 4 \end{pmatrix} = \begin{bmatrix} 1.3 \\ 1.4 \end{bmatrix}$ and repeat

Let's translate this ~~fitting~~ $\mathbb{D}$ to ~~fitting $\mathbb{D}$~~ an ~~learning~~.

Gradient Descent

$f(\vec{x}) \longrightarrow$ objective function on the current fit $\vec{\hat{y}}$, $L(\vec{y}, \vec{\hat{y}})$ which must be differentiable in all n dimensions

$\vec{x}_0 \longrightarrow$ Starting prediction $\vec{\hat{y}}_0$

$$\vec{\nabla} f(\vec{x}) \rightarrow \vec{\nabla} L(\vec{y}, \vec{\hat{y}}) = \begin{bmatrix} \frac{\partial L}{\partial \hat{y}_1} \\ \vdots \\ \frac{\partial L}{\partial \hat{y}_n} \end{bmatrix}$$

$$\vec{\hat{y}}_1 = \vec{\hat{y}}_0 - \eta \vec{\nabla} L(\vec{y}, \vec{\hat{y}}_0)$$

$$\vec{\hat{y}}_2 = \vec{\hat{y}}_1 - \eta \vec{\nabla} L(\vec{y}, \vec{\hat{y}}_1)$$

$$\vdots$$

$\vec{\hat{y}}_m \approx \vec{\hat{y}}_*$, the predictions that minimize the objective function.

But the goal of supervised learning is <u>not</u> to find best $\vec{\hat{y}}$ for $\mathbb{D}$, it's to build a generalizable model g.

So we instead want to use gradient descent in the space of functions $g: \mathbb{R}^p \rightarrow \mathbb{R}$. How can we amend the above to do that? This is why it took until 2001 to invent this as it's not so easy to "port" this idea over.

the trick is to fit a model to the gradient itself.
This is strange as we've only fit models to data. So
let's translate:

Fitting ① via Gradient Descent		Gradient boosting for supervised learning using base learner $\mathcal{A}$
$\hat{y}_0$	$\longrightarrow$	g_0, the default model
$\hat{y}_1 = \hat{y}_0 - \eta \vec{\nabla} L(\vec{y}, \hat{y}_0)$	$\longrightarrow$	$G_1 = g_0 + \eta \mathcal{A}(\langle X, -\vec{\nabla} L(\vec{y}, \vec{\hat{y}}_0)\rangle, \mathcal{H})$

$g_0(X)$ (= $\hat{y}_0$)

instead of fitting a model to the data, you're fitting a model to the negative gradient

Let $G_t := g_0 + g_1 + \dots + g_t$

the partial sums

g_1 is a step in the direction in <u>function</u> space towards the "best" model

$$\hat{y}_2 = \hat{y}_1 - \vec{\nabla}(\vec{y}, \hat{y}_1) \longrightarrow G_2 = G_1 + \underbrace{\eta \mathcal{A}(\langle X, -\vec{\nabla}(\vec{y}, \vec{\hat{y}}_1)\rangle, \mathcal{H})}_{g_1}$$

$G_1(X)$ (= $\hat{y}_1$)

$\vdots$

$$\hat{y}_M = \hat{y}_{M-1} - \vec{\nabla}(\vec{y}, \vec{\hat{y}}_{M-1}) \to G_M = G_{M-1} + \underbrace{\eta \mathcal{A}(\langle X, -\vec{\nabla}(\vec{y}, \hat{y}_{M-1})\rangle, \mathcal{H})}_{g_M}$$

$G_{M-1}(X)$ (= $\hat{y}_{M-1}$)

Which algorithms work well in gradient boosting? Weak learners (high bias, low variance) that can still "span" the space of non-linear and interactive models (interactions among the p features).

$\Longrightarrow$ trees with low depth / high N_0 node size is usually the default choice

We will now apply gradient boosting to both regression and prob. estimation for binary outcome.

Regression: let the objective function to be minimized be:

$$L(\vec{y}, \hat{\vec{y}}) := \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = SSE.$$ Doesn't have to be... just must be differentiable

$$-\vec{\nabla} L(\vec{y}, \hat{\vec{y}}) = \begin{bmatrix} -2(y_1 - \hat{y}_1) \\ -2(y_2 - \hat{y}_2) \\ \vdots \\ -2(y_n - \hat{y}_n) \end{bmatrix} = -2\vec{e}$$

$\Rightarrow$ essentially you fit a shallow ~~regression tree~~ to $\langle X, \vec{e} \rangle$ modulo a constant... very intuitive

Prob est: recall the likelihood function

$$\prod_{i=1}^{n} \hat{p}_i^{\,y_i} (1-\hat{p}_i)^{1-y_i}$$

It will be easier to use log odds:

$$\text{let } \hat{y}_i := \ln\left(\frac{\hat{p}_i}{1-\hat{p}_i}\right) \underset{\substack{1:1 \\ \text{function}}}{\Longleftrightarrow} \hat{p}_i = \frac{e^{\hat{y}_i}}{1+e^{\hat{y}_i}}$$

The likelihood function becomes

$$\prod_{i=1}^{n} \left(\frac{e^{\hat{y}_i}}{1+e^{\hat{y}_i}}\right)^{y_i} \left(\frac{1}{1+e^{\hat{y}_i}}\right)^{1-y_i} = \prod_{i=1}^{n} \frac{e^{y_i \hat{y}_i}}{1+e^{\hat{y}_i}}$$

Since maximizing likelihood is the same as max. log likelihood, we can maximize

$$\ln(\ \ ) = \sum_{i=1}^{n} \ln\left(\frac{e^{y_i \hat{y}_i}}{1+e^{\hat{y}_i}}\right) = \sum_{i=1}^{n} y_i \hat{y}_i - \ln(1+e^{\hat{y}_i})$$

Since we want to minimize functions in gradient descent, we now have an objective function:

$$L(\vec{y}, \hat{\vec{y}}) = \sum_{i=1}^{n} -y_i \hat{y}_i + \ln(1+e^{\hat{y}_i})$$

$$-\vec{\nabla} L(\vec{y}, \hat{\vec{y}}) = \begin{bmatrix} -y_1 + \frac{e^{\hat{y}_1}}{1+e^{\hat{y}_1}} \\ -y_2 + \frac{e^{\hat{y}_2}}{1+e^{\hat{y}_2}} \\ \vdots \\ -y_n + \frac{e^{\hat{y}_n}}{1+e^{\hat{y}_n}} \end{bmatrix} = -\vec{y} + \frac{e^{\hat{\vec{y}}}}{1+e^{\hat{\vec{y}}}} \overset{\text{vectorwise}}{=} -\vec{y} + \hat{\vec{p}}$$

essentially you fit a shallow ~~regression~~ tree to $\langle X, \vec{y} - \hat{\vec{p}} \rangle$ modulo a cons... intuitive!!

↑ end of course

↓ extra

Possible improvement: eliminate hyperparameter η.

$$\text{let } G_t = G_{t-1} + \eta^* \underbrace{\mathcal{A}\left(\langle X, -\vec{\nabla}(\vec{y}, G_{t-1})\rangle, \mathcal{H}\right)}_{\tilde{g}_t}$$

$$\text{where } \eta^* = \underset{\eta \in (0,\infty)}{\text{argmin}} \left\{ L\left(\vec{y}, G_{t-1}(X) + \eta\, \tilde{g}_t(X)\right) \right\}$$