Gradient boosting

- can be used for classification or regression
- iterative method (not easily parallelizable)
- ensemble model
- need to choose an appropriate loss function given problem at hand
- need to choose a set of weak learners to choose from at each iteration, $\mathcal{H}$ (often trees/stumps can also be splines or anything else.
- the algo learns a function of the form:

$$F(x) \in \left\{ \sum_{m=1}^{M} \nu_m h(x; a_m) \;\middle|\; \nu_m \in \mathbb{R}, h(\cdot) \in \mathcal{H} \right\}$$

idea of gradient boosting

- instead of fitting a parameterized model by employing GD in parameter space to minimize a cost function, run gradient decent in a function space of dims $N \leftarrow$ # data points ($\mathbb{R}^N$), and at each iteration approximate the gradient by choosing a parameterized weak learner from $\mathcal{H}$.

• goal is to find:

$$F^* = \underset{F \in \mathcal{F}}{\arg\min} \left\{ \sum_{i=1}^{N} l(y^{(i)}, F(x^{(i)})) \right\}$$ (i.e. minimize empirical risk)

let $\vec{F} \doteq [F(x^{(1)}), F(x^{(2)}), \dots, F(x^{(N)})]^T$

$$\Rightarrow J(\vec{F}) = \sum_{i=1}^{N} l(y^{(i)}, \vec{F}_i)$$

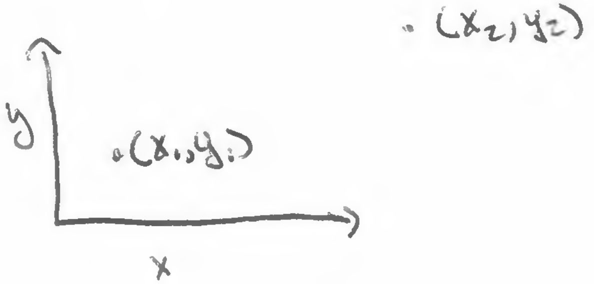
$\Rightarrow$ thus we run GD on the space formed by $F_1, F_2, \dots, F_N$ to find $\vec{F}^*$ the vector that minimizes the cost

$$\vec{F}^{(m)} = \vec{F}^{(m-1)} - \rho_m \vec{g}_m$$

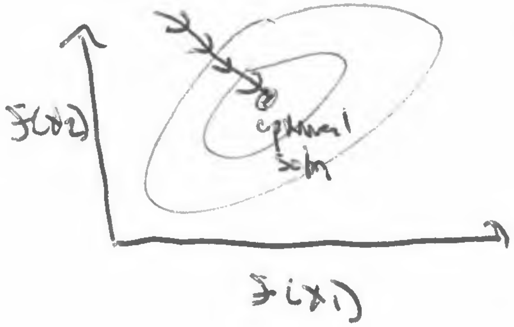
where $$[\vec{g}_m]_j = \frac{\partial}{\partial F_j} \sum_i l(y^{(i)}, F_i^{(m-1)}) = \frac{\partial l(y^{(j)}, F_j^{(m-1)})}{\partial F_j}$$

and $$\rho_m = \underset{\rho \in \mathbb{R}}{\arg\min} \left\{ \sum_i l(y^{(i)}, \vec{F}^{m-1} - \rho \vec{g}_m) \right\}$$

- in pictures, if we have 2 data points in the training set:



$\Rightarrow$



- Note that we wish to predict y at more points than just x_1, x_2
- to extrapolate to other values of x, we guess a parameterized function for the gradient at each boosting iteration. Let $\vec{\phi}_{\vec{a}} \equiv [\phi(x_1; \vec{a}), \phi(x_2; \vec{a}) \dots \phi(x_N; \vec{a})]^T$

Then the algo is:

① $F^0(x) = 0$

② For $m=1$ to M:

- compute $\vec{g}_m$ as given by formula on previous page
- approximate $\vec{g}_m$ w/ a parameterized weak learner by finding the vector closest in $\mathbb{R}^N$ (in L^2 sense)

$$\phi(x, a_m) = \underset{a}{\operatorname{argmin}} \left\{ \sum_{i=1}^{N} \left(-\vec{g}_m^{(i)} - \vec{\phi}_{\vec{a}}^{(i)} \right)^2 \right\}$$

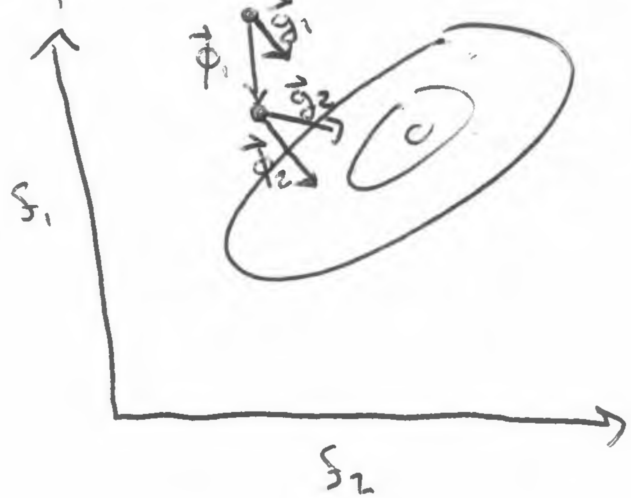
$\rightarrow$

- $\rho_m = \underset{\rho}{\arg\min} \left\{ \sum_i l(y^{(i)}, \vec{F}^{m-1} - \rho\, \phi(x^{(i)}, \vec{a}_m)) \right\}$
- $F^m := F^{m-1} + \rho_m \phi(x; a_m)$

output: F^M

- For regression: $\hat{y}(x) = \sum_{m=1}^{M} \rho_m \phi(x; a_m)$ (a weighted sum)
- For classification: $\hat{y} = \text{sign}\left(\sum_{m=1}^{M} \rho_m \phi(x; a_m) \right)$ (loss function typically defined in such a way that it is easy to compute class probabilit from $\sum \rho_m \phi(x, a_m)$)

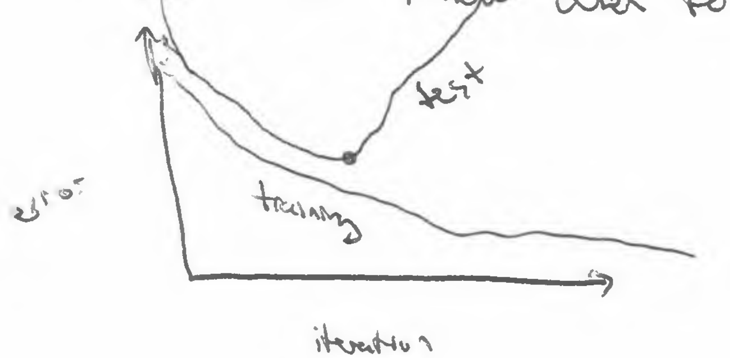
in pictures:



- thus we traverse "F" space like this

- ways to regularize/improve test set performance
  - shrinkage let $\nu \in [0,1]$ ← step size
  - by making step size smaller test error seems to improve (will need more iterations to converge though)

- hold out some of the data on each iteration (fit on less training data, so less propensity to overfit)
  - has the advantage that we can test error at each iteration on an oob sample to build up a learning curve to know when to stop



- early stopping

★ these regularization methods seem to work well from empirical studies.

Pros

- typically very good performance
- lots of flexibility in choosing loss function + weak learner to deal w/ problem at hand
- can learn complicated decision functions
- non-parametric (don't need to have much knowledge about how function should look)

Cons

- can't parallelize (XGBoost can ||ize trees, but not Ξ trees k)
- can be difficult to tune
- can overfit if not careful
- training can take a while