CS 559: Random Forests and Boosting Methods

Lecture 7

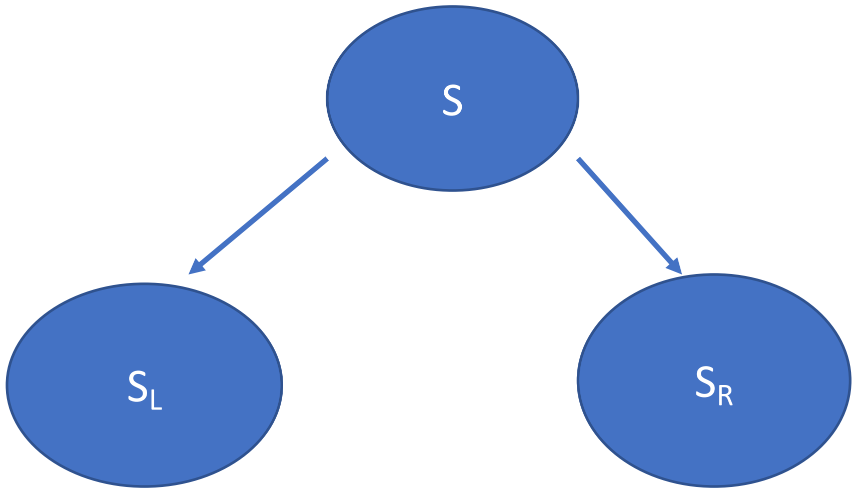
Announcement

- Midterm Exam is done.
- Spring break next week.
- New assignment will be published after the spring break.
- So as the project.

Outline

- Decision Trees Review
- Bagging
- Random Forest
- Gradient Boost
- AdaBoost

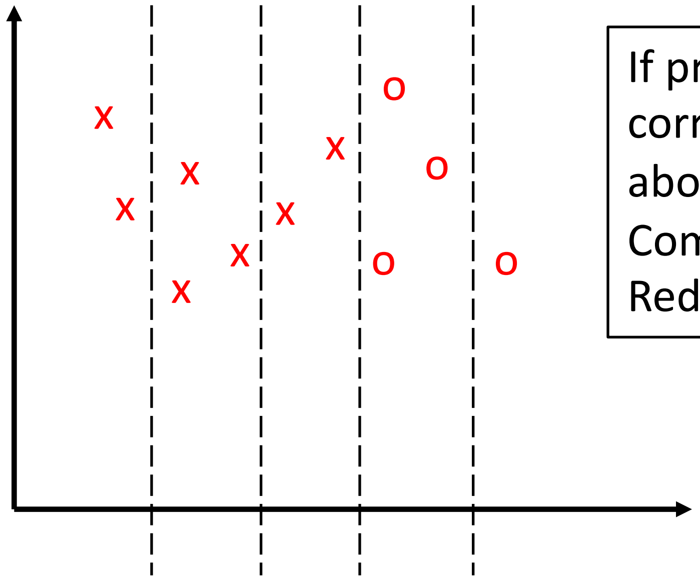
Last Lecture: Decision Trees - Classification



Impurity of Split: $\frac{|S_L|}{|S|}H(S_L) + \frac{|S_R|}{|S|}H(S_R)$

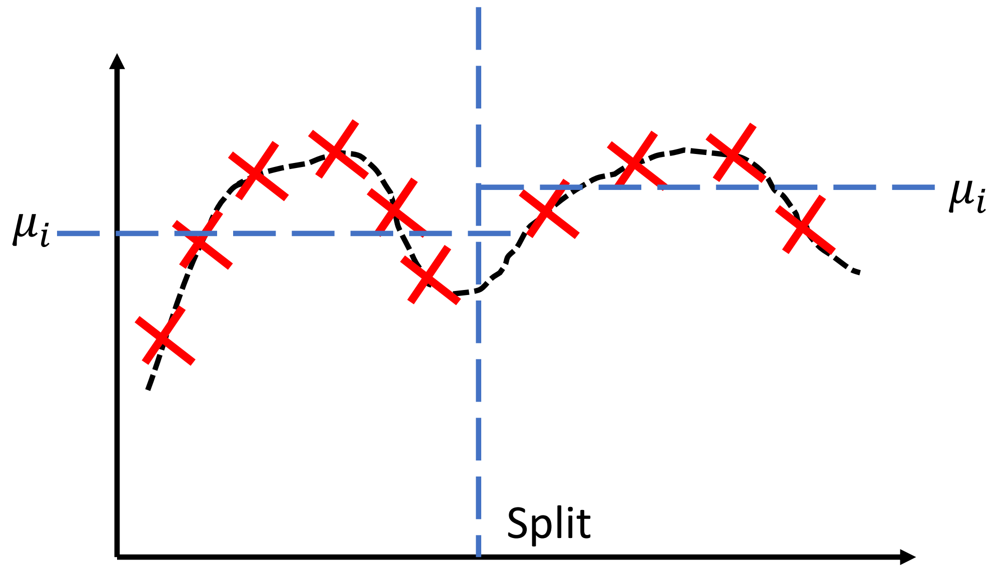
Computation Complexity: $\underbrace{ND}_{\text{\# of Split}}\times \underbrace{NK}_{\text{\# of Class}} = DKN^2$

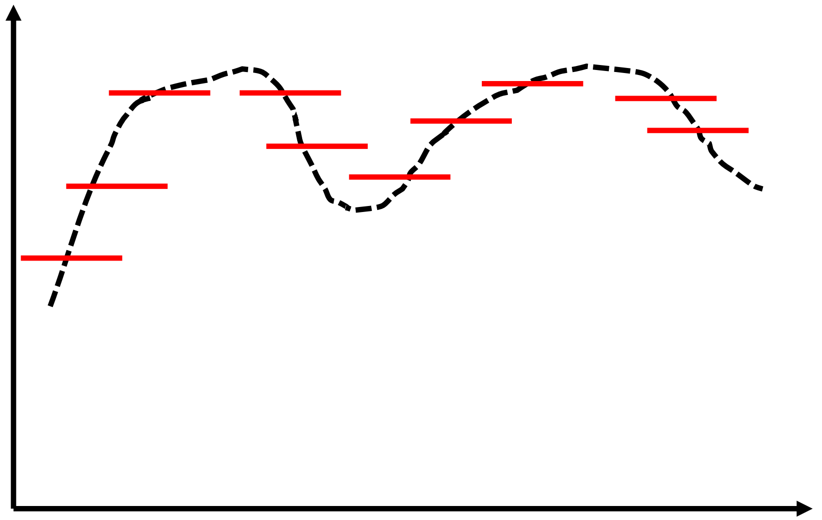
Any ways to reduce the computation complexity?
How about splitting one by one from the previous split?



If previous label is already classified correctly, we then can just worry about the next label.
Computation: NDK
Reduced from quadratic!

Last Lecture: Decision Trees - Regressions





Loss Function: $l(S) = \frac{1}{|S|}\sum_{(x,y)\in S}(y-\mu)^2$ where $\mu = \frac{1}{|S|}\sum y$

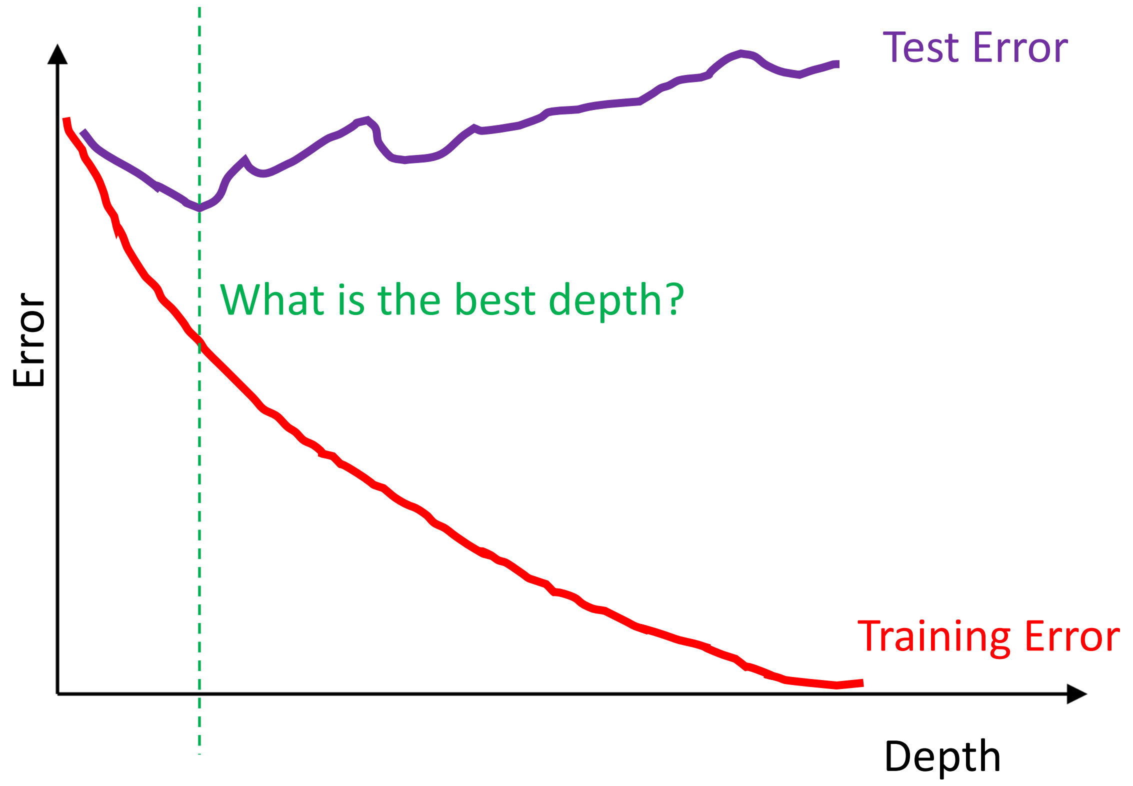
Goal: how to get to close to μ

But: becomes variance $(y-\mu)^2$ problem. Balancing the bias-variance tradeoff is the key!

- Limiting the depth of tree: Bias vs. Variance
- # of leaves: each leaf predicts one value - smoothness

We can use any loss functions

Last Lecture: Decision Trees



<u>Limitation of Decision Trees</u>
Increasing depth -> Over-fitting
Node is an integer -> minimum test error not occur at the exact node.

We can use bagging method!

Bagging: Bootstrap Aggregating

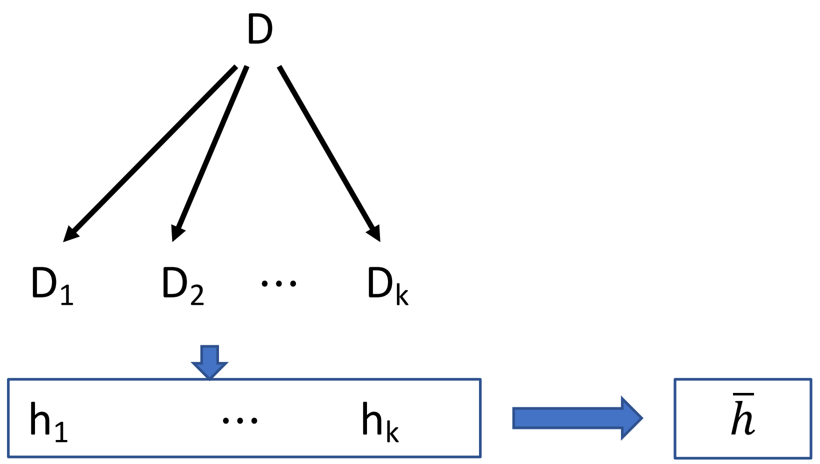
- Take repeated bootstrap samples from training set D (Breiman, 1994)
- Bootstrap sampling: Given set D containing N training examples, create D' by drawing N examples **at random with replacement** from D
- Bagging:
  - Create k bootstrap samples $D_1, \dots, D_k$
  - Train distinct classifier on each D_i
  - Classify new instances by majority vote/average

$$h(x) = \frac{1}{k}\sum_{j=1}^{k} h_{D_j}(x) \xrightarrow{k\to\infty} \bar{h}(x)$$

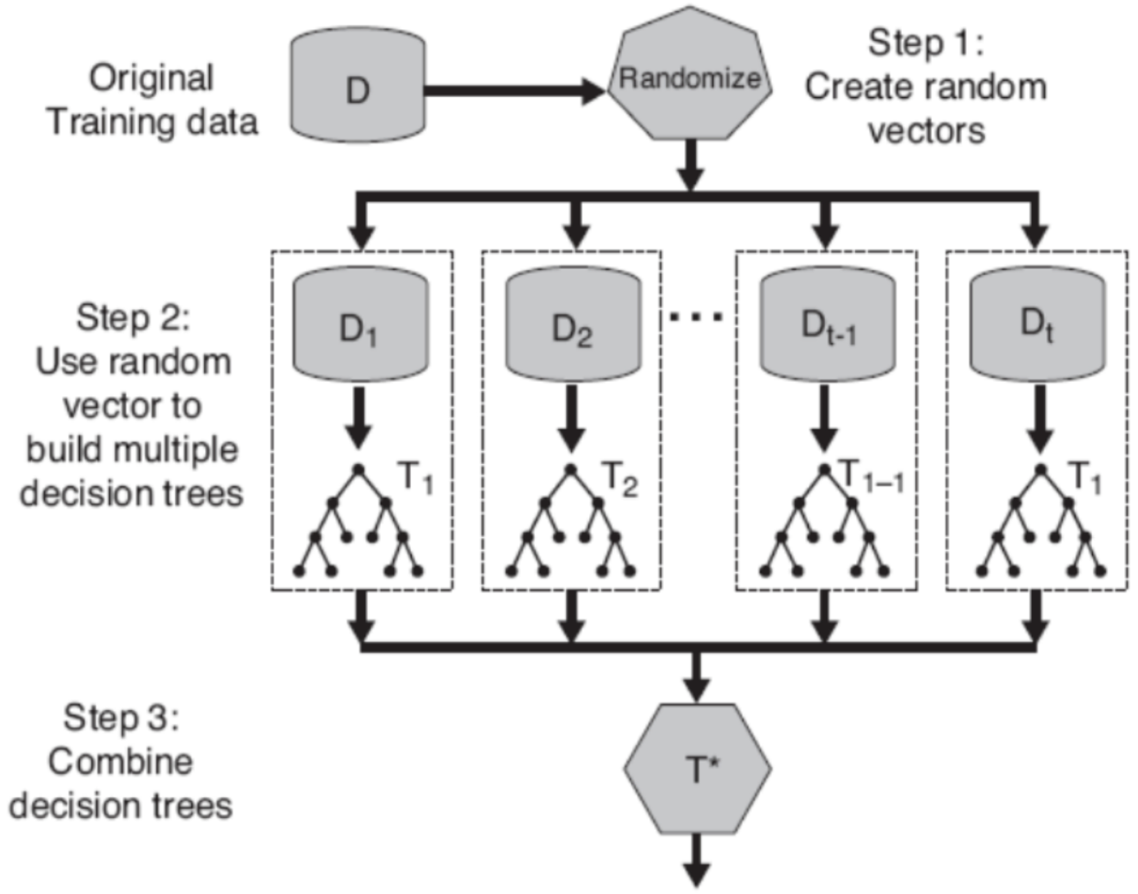
  - Goal: Reduce the variance $E\left[\left(h_D(x) - \bar{h}(x)\right)^2\right]$

Bagging: Bootstrap Aggregation

- To ensure diverse classifiers, the base classier should be unstable, that is, small changes in the training set should lead to large changes in the classier output.
- Large error reductions have been observed with decision trees and bagging. This is because decision trees are highly sensitive to small perturbations of the training data.
- Bagging is not effective with nearest neighbor classifiers. NN classifiers are highly stable with respect to variations of the training data.
- When the errors are highly correlated, and bagging becomes ineffective.



Random Forests


Original Training data
D
Randomize
Step 1: Create random vectors
Step 2: Use random vector to build multiple decision trees
D1
D2
...
Dt-1
Dt
T1
T2
T1–1
T1
Step 3: Combine decision trees
T*

Ensemble method specifically designed for decision tree classifiers.

Two sources of randomness: “bagging” and “random input vectors”

Use bootstrap aggregation to train many decision trees.

- Randomly subsample n examples
- Train decision tree on subsample
- Use average or majority vote among learned trees as prediction

Also randomly subsample features: best split at each node is chosen from a random sample of m attributes instead of all attributes

Random Forests - Algorithm

For b = 1 to B

- Draw a bootstrap sample of size N from the data D with k attributes.
- Grow a tree T_b using the bootstrap sample as follows
  - Choose m attributes (m< all attributes) uniformly at random from the data
  - Choose the best attribute among the m to split on
  - Split on the best attribute and recurse until partitions have fewer than s_{min} number of nodes
- Prediction for a new data point x
  - Regression: $\frac{1}{B}\sum_b T_b(x)$
  - Classification: choose the majority class label among $T_1(x), \ldots, T_B(x)$

1. Split each training set into two partitions, P and Q, to make the classifier consistent.
2. Do not grow tree to end. Instead, prune based on the leave out sample.

The Boosting Approach

Bagging reduces variance by averaging but has little effect on bias.
Can we average and reduce bias? (Michael Kerns in 1988)

- Yes, Boosting! (Robert Schapire in 1990)

- devise computer program for deriving rough rules (weak classifier)
- apply procedure to subset of examples and obtain a simple rule
- apply to 2nd subset of examples and obtain a 2nd rule
- repeat T times

How to choose examples on each round?

- concentrate on “hardest” examples (those most often misclassified by previous rule)

How to combine the rules into single prediction rule?

- take (weighted) majority vote of rules

boosting = general method of converting rough rules into highly accurate prediction rule
technically

- assume given “weak” learning algorithm that can consistently find classifiers at least slightly better than random, say, accuracy 55%
- given sufficient data, a boosting algorithm can provably construct single classier with very high accuracy say, 99%

Boosting - gradient descent in function space

Let $\mathcal{H}$ be hypothesis class and H be the ensemble classifier,

$$l(H) = \frac{1}{n}\sum_{i=1}^{n} l(H(x_i), y_i)$$

where $H(x) = \sum_{t=1}^{T} \alpha h_t(x)$ and $h_{t+1} = argmin_{h\in\mathcal{H}} l(H_t + \alpha h_t)$.

Once h_{t+1} is found, add to the ensemble $H_{t+1} = H_t + \alpha h_{t+1}$.

Taylor Approximation.
Constant and can be ignored when we minimize it.

Inner product

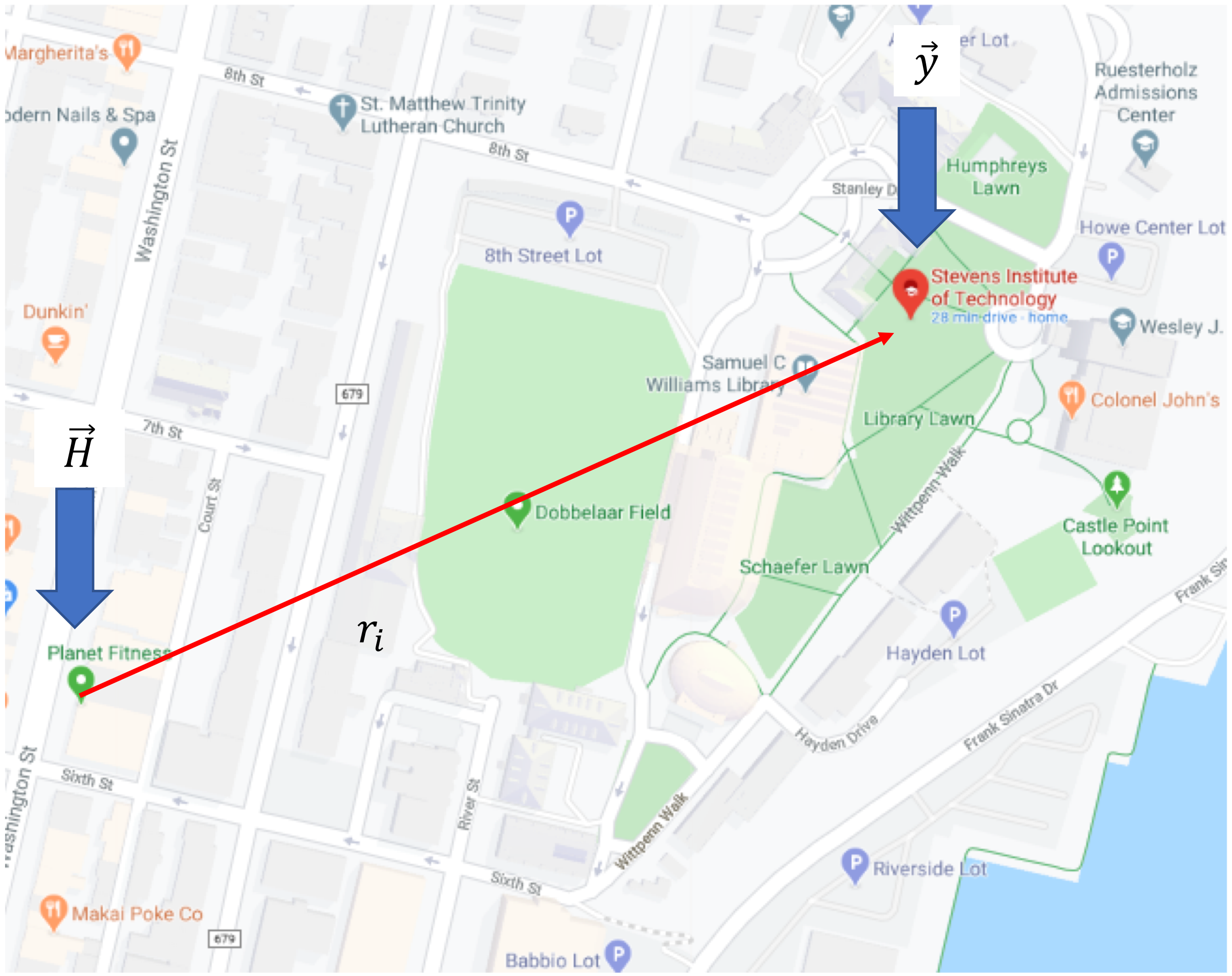
$$l(H + \alpha h) \approx l(H) + \alpha \overbrace{< \nabla l(H), h >}$$
$$argmin_{h\in H} l(H + \alpha h) \approx argmin_{h\in H} < \nabla l(H), h >$$

$$= argmin_{h\in\mathcal{H}} \sum_{i=1} \frac{\partial l}{\partial H(x_i)} h(x_i)$$

We can do the boosting if we have an algorithm that solves as long as

$$h_{t+1} = argmin_{h\in H} \sum_{i=1} \underset{\underset{r_i}{\parallel}}{\frac{\partial l}{\partial H(x_i)}} h(x_i) < 0$$

Boosting – general pseudo code



```
$H = 0$
for $t = 1 : T - 1$ do
    $r_i = \frac{\partial l((H_t(x_1), y_1), \dots, (H_t(x_n), y_n))}{\partial H(x_i)}$
    $h_{t+1} = argmin_{h \in \mathcal{H}} \sum_{i=1}^{n} r_i h(x_i)$
    if $\sum_{i=1}^{n} r_i h_{t+1}(x_i) < 0$ then
        $H_{t+1} = H_t + \alpha_{t+1} h_{t+1}$
    else
        return $H_t$
    end
end
return $H_T$
```

Boosting – Gradient Boost

Classification & Regression
Weak learners, $h \in \mathcal{H}$, are regressors $h(\boldsymbol{x}) \in \mathcal{R}, \forall \boldsymbol{x}$, typically fixed-depth (between 4-6) regression trees.
Step size α is fixed to a small constant.
Loss function: Any differentiable convex that decomposes over the sample

$$\mathcal{L}(H) = \sum_{i=1}^{n} l\big(H(\boldsymbol{x_i})\big)$$

Must to find a tree $h()$ that maximizes

$$h = argmin_{h \in \mathcal{H}} \sum_{i=1}^{n} \underbrace{\frac{\partial l}{\partial H(\boldsymbol{x_i})}}_{\substack{\parallel \\ r_i}} h(\boldsymbol{x_i})$$

Boosting – Gradient Boost

Assumptions:

1. $\sum_{i=1}^{n} h^2(x_i) = Constant$ – simple to normalize the predictions and important since we can always decrease $\sum_{i=1}^{n} h(x_i) r_i$ by rescaling h with a large constant.
2. CART trees are closed.
3. Define the negative gradient as $t_i = -r_i$.

3: we can square this as this does not have nothing to do with *h()*

$$argmin_{h \in \mathcal{H}} \sum_{i=1}^{n} r_i h(x_i) = -argmin_{h \in \mathcal{H}} \sum_{i=1}^{n} t_i^2 - 2 t_i h(\boldsymbol{x_i}) + \left(h(\boldsymbol{x_i})\right)^2$$

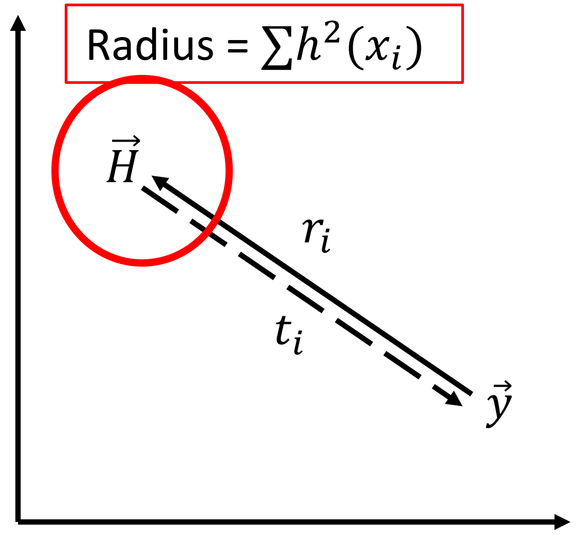
1: Enforcing interested only in direction

$$= -argmin_{h \in \mathcal{H}} \sum_{i=1}^{n} (h(\boldsymbol{x_i}) - t_i)^2$$

2: minimizing is half is good as twice.

We can use the good old regression trees.

Radius = $\sum h^2(x_i)$



Boosting – Gradient Boost

If the loss is a square loss, $l(H) = \frac{1}{2}\sum_{i=1}^{n}(H(x_i) - y_i)^2$,

$$t_i = -\frac{\partial l}{\partial H(x_i)} = y_i - H(\boldsymbol{x_i})$$

It is a residual.

We can use any differentiable and convex loss function and the solution for the next weak learner will always be the regression tree minimizing the square loss.

Pseudo-Code

```
$H = 0$
for $t = 1:T$ do
    $t_i = y_i - H(x_i)$
    $h = argmin_{h \in \mathcal{H}}(h(x_i) - t_i)^2$
    $H = H + \alpha h$
end
return H
```

Boosting - AdaBoost

- Classification ($y_i \in \{+1, -1\}$)
- Weak learners, $h \in \mathcal{H}$ are binary, $h(x_i) \in \{-1, +1\}, \forall x$
- We perform line-search to obtain best step-size α
- Exponential loss $l(H) = \sum_{i=1}^{n} e^{-y_i H(x_i)}$

- The gradient is $r_i = -y_i e^{-y_i H(x_i)}$
- Notations:
  - Let $w_i = \frac{e^{-y_i H(x_i)}}{Z}$ where z is the normalizing factor $Z = \underbrace{\sum_{i=1}^{n} e^{-y_i H(x_i)}}_{\text{Exponential loss function}}$.
  - This makes $\sum_{i=1}^{n} w_i = 1$ and w_i is the weight.
- The next weak learner can be solved by optimization.

$$h(x_i) = argmin_{h \in \mathcal{H}} \sum_{i=1}^{n} -y_i \underbrace{e^{-y_i H(x_i)}}_{\text{Substitute in } w_i} h(x_i) = argmin_{h \in \mathcal{H}} \sum_{i=1}^{n} - \underbrace{y_i w_i h(x_i)}_{y_i h(x_i) \in \{-1, +1\}} = argmin_{h \in \mathcal{H}} \sum_{h(x_i) \neq y_i} w_i + \underbrace{\sum_{h(x_i) = y_i} w_i}_{1 - \sum_{h(x_i) \neq y_i} w_i}$$

$$= argmin_{h \in \mathcal{H}} \sum_{h(x_i) \neq y_i} w_i \Leftarrow \boxed{\text{The weighted classification error, } \epsilon < 0.5}$$

Boosting - AdaBoost

- We can find the optimal step-size in the closed form every time we take a "gradient" step.
- With given l, H, h

$$\alpha = argmin_{\alpha} l(H + \alpha h) = argmin_{\alpha} \sum_{i=1}^{n} e^{-y_i[H(x_i)+\alpha h(x_i)]}$$

- Differentiate respect to α and set to 0:

$$0 = \sum_{i=1}^{n} y_i h(x_i) e^{-y_i[H(x_i)+\alpha h(x_i)]} = -\sum_{h(x_i)y_i=1} e^{-y_i[H(x_i)+\alpha h(x_i)]} + \sum_{h(x_i)y_i=-1} e^{-y_i[H(x_i)+\alpha h(x_i)]}$$

$\boxed{y_i h(x_i) \in \{-1, +1\}}$

$\boxed{\text{Substitute in } w_i}$ $\boxed{\text{Substitute in } w_i}$

$$= -\sum_{h(x_i)y_i=1} w_i e^{-\alpha} + \sum_{h(x_i)y_i=-1} w_i e^{\alpha} \Rightarrow -(1-\epsilon)e^{-\alpha} + \epsilon e^{\alpha} = 0$$

$\boxed{\text{Weighted error}}$

$$\Rightarrow \alpha = \frac{1}{2} \ln \frac{1-\epsilon}{\epsilon}$$

Boosting - AdaBoost

- After a taking a step, we need to re-compute all the weights and then re-normalize.
- Let the unnormalized weight be $\hat{w}_i$

$$\hat{w}_i \leftarrow \hat{w}_i e^{-\alpha h(x_i) y_i}$$

- The normalizer Z becomes

$$Z \leftarrow Z\left(2\sqrt{\epsilon(1-\epsilon)}\right)$$

- Then

$$w_i \leftarrow \frac{w_i e^{-\alpha h(x_i) y_i}}{2\sqrt{\epsilon(1-\epsilon)}}$$

```
Pseudo-Code:
$H_0 = 0$ & $w_i = \frac{1}{n}, \forall i$
for $t = 0: T - 1$ do
    Calculate h & $\epsilon$
    if $\epsilon < 0.5$ then
        calculate $\alpha, H_{t+1}, w_i \forall i$
    else
        return $H_t$
    end
end
return $H_T$
```

Summary

- Decision Trees: need to reduce variance. How?
- Bagging: Bootstrap (random subsampling with replacement)
- Random Forest
  - Bagging method with full decision tree method
  - Easy, feature selection, less data pre-processing
  - But… How to reduce bias?
- Boosting
  - Gradient Boost
    - Good for classification & regression
    - Simple when we use the square loss function
    - Constant small step-size
    - Works with any convex differentiable loss function
  - AdaBoost
    - Only for classification
    - Invented first but turned to be one of gradient boost (exponential loss function)
    - Need to compute weight and step-size for every iteration