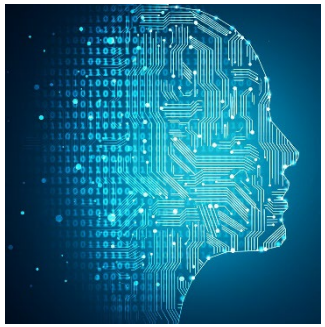
Machine Learning

Ensemble Methods

Kevin Moon (kevin.moon@usu.edu)

STAT/CS 5810/6655



Outline

1. Motivation
2. Example: Average shifted histograms
3. Bagging
4. Random Forests
5. Boosting
6. XGBoost

Motivation

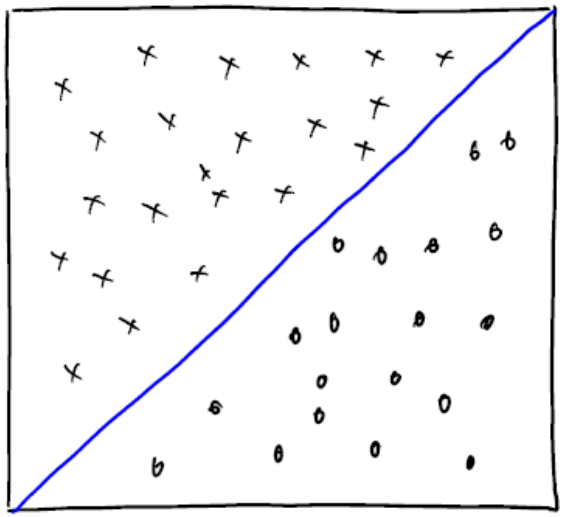
- Consider a classification problem
- Let $f_1, \ldots, f_T$ be a set of different classifiers for this problem
- Idea: combine these classifiers into a single classifier that outperforms any individual classifier
- Let's consider a simple example

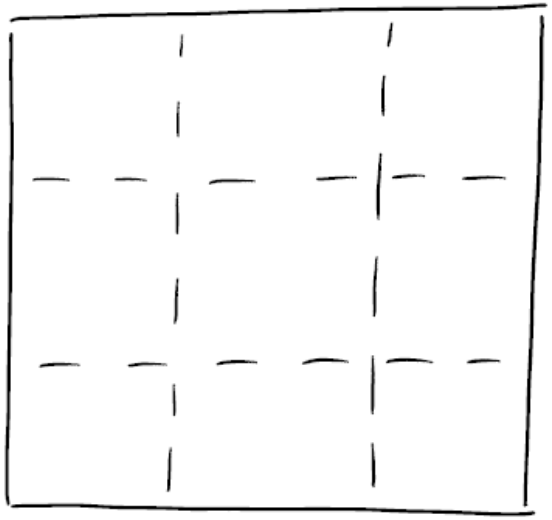
Averaged Shifted Histograms

- Suppose feature space is $[0,1]^2$
- Bayes risk/error is zero
- Marginal of $\boldsymbol{X}$: uniform
- Marginal of Y: uniform

- Suppose we are using histogram classifiers
  - Partitions the space ($[0,1]^2$ in this case) into squares
  - Label of each square determined by *majority vote*

Example of data realizations



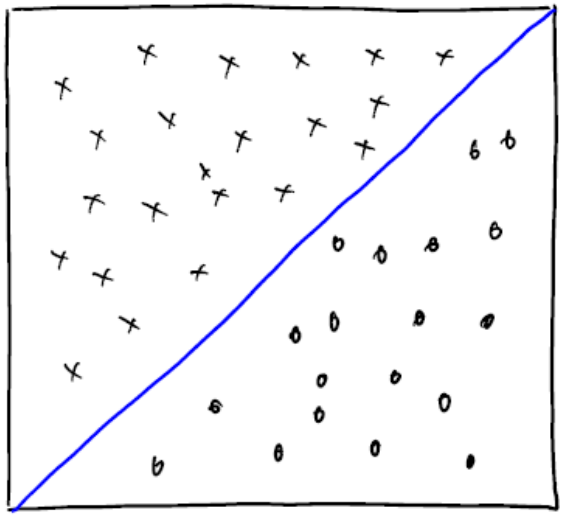


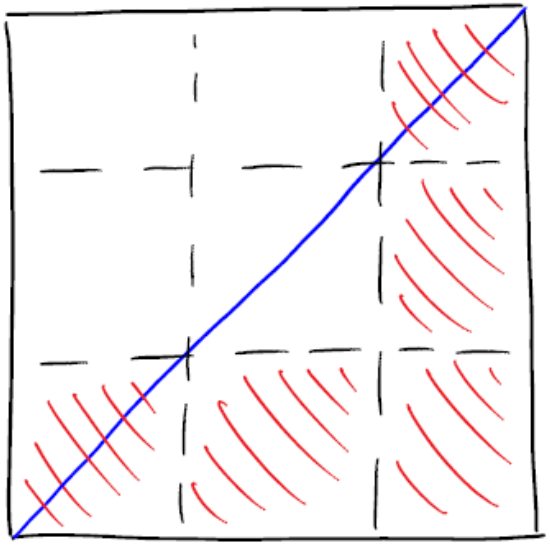
9 square partition

Averaged Shifted Histograms

- This classifier will not perform very well for the given distribution
  - Or most distributions
- A third of the time, the classifier has a 50/50 chance of being wrong
  - Risk $= \frac{1}{3} \cdot \frac{1}{2} = \frac{1}{6}$
- However, we can do much better with an appropriate ensemble method

Example of data realizations





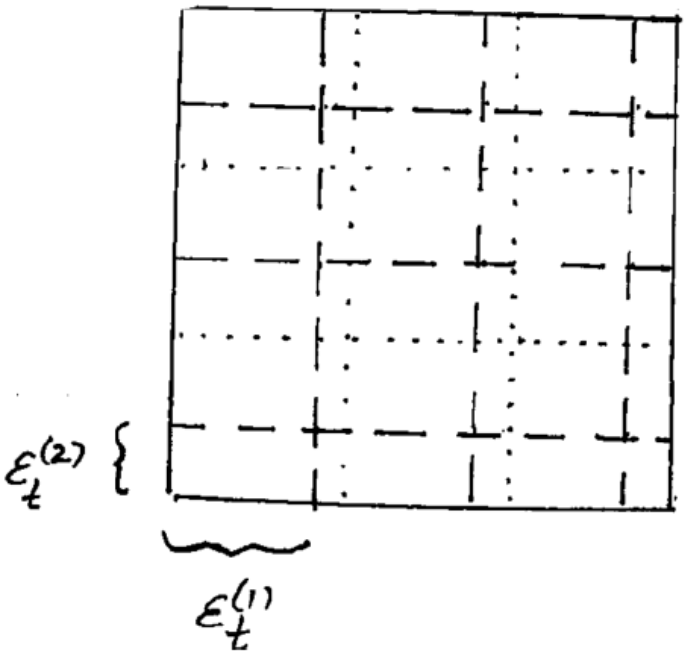
9 square partition

Averaged Shifted Histograms

- Try the following:

- For $t = 1, \dots, T$
  - Generate $\epsilon_1^{(t)}, \epsilon_2^{(t)} \in \left[0, \frac{1}{3}\right)$ uniformly at random
  - Shift the partition by $\left[\epsilon_1^{(t)}, \epsilon_2^{(t)}\right]^T$ and construct f_t based on the shifted partition
- Output $f(\boldsymbol{x})$ = majority vote over $f_1(\boldsymbol{x}), \dots, f_T(\boldsymbol{x})$
- The ensemble classifier performs remarkably well even though each individual classifier performs poorly

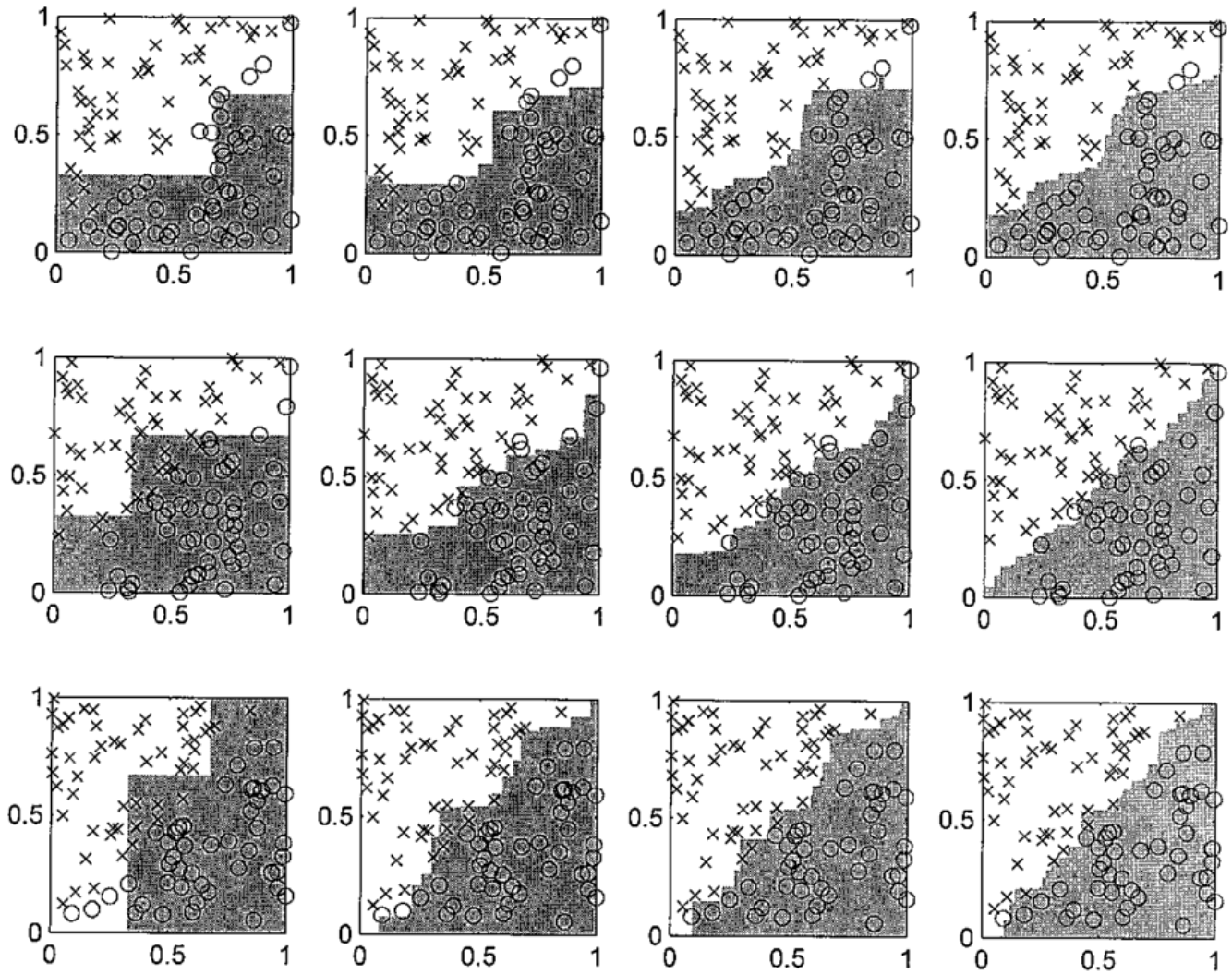
A single randomly shifted histogram classifier



Average Shifted Histograms

of classifiers = 1 5 11 21

3 realizations of data

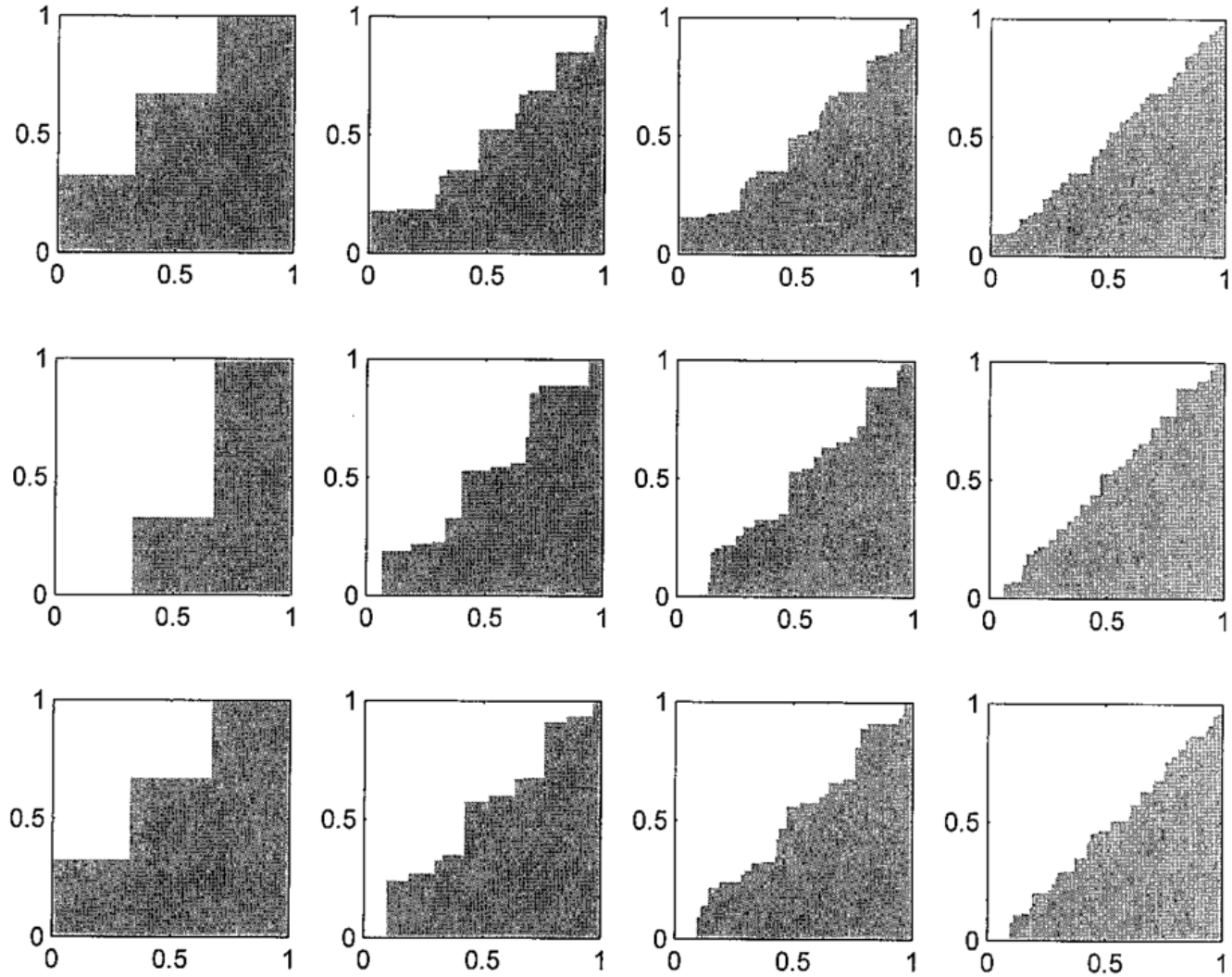


$n = 100$

Average Shifted Histograms

of classifiers = 1 5 11 21

3 realizations of data



$n = 1000$

Stability

- An ML method is *stable* if a small change in the input leads to a small change in the output (e.g. the learned classifier)

- **Example**: decision trees are unstable classifiers
  - They benefit considerably from ensemble methods

- Most ensemble methods result from introducing some form of randomization into the learning algorithm

Bagging

- Bagging = *bootstrap aggregation*
  - Developed by Leo Breiman
- Let $B \geq 1$ be an integer.
- Given a training sample of size n, let I_b for each $b = 1, \ldots, B$ be a list of size n obtained by sampling from $\{1,2, \ldots, n\}$ *with replacement*
- I_b is called a *bootstrap sample*

Bagging

- Now suppose we have a fixed learning strategy
  - E.g. decision trees
- Let f_b be a classifier obtained by applying this strategy to $\{(\boldsymbol{x}_i, y_i)\}_{i \in I_b}$
- The bagging classifier is

$$f(\boldsymbol{x}) = \text{majority vote over } f_1(\boldsymbol{x}), \dots, f_B(\boldsymbol{x})$$

- Bagging has been shown to improve the performance and stability of decision trees

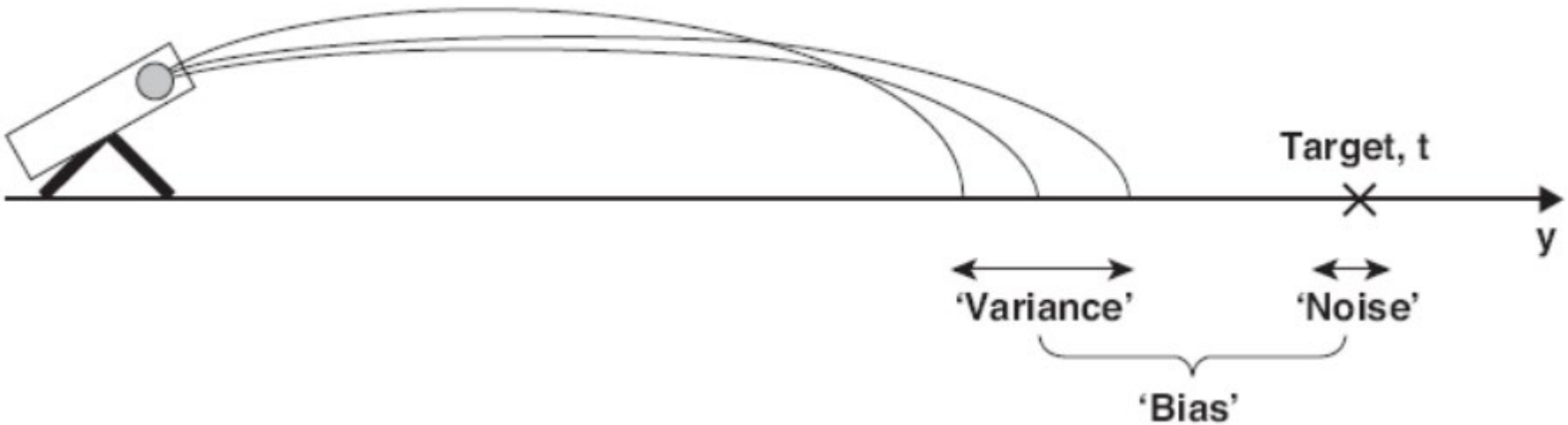
Random Forests

- Random forest = an ensemble of decision trees where each decision tree is randomized in an i.i.d. fashion
- **Example**: Bagging with decision trees is an example of a random forest
- Problem with this example: bootstrap samples are highly correlated (we're sampling with replacement)
  - Different decision trees tend to select the same informative features ⇒ highly correlated predictions
  - Is this a problem?

Bias-variance tradeoff

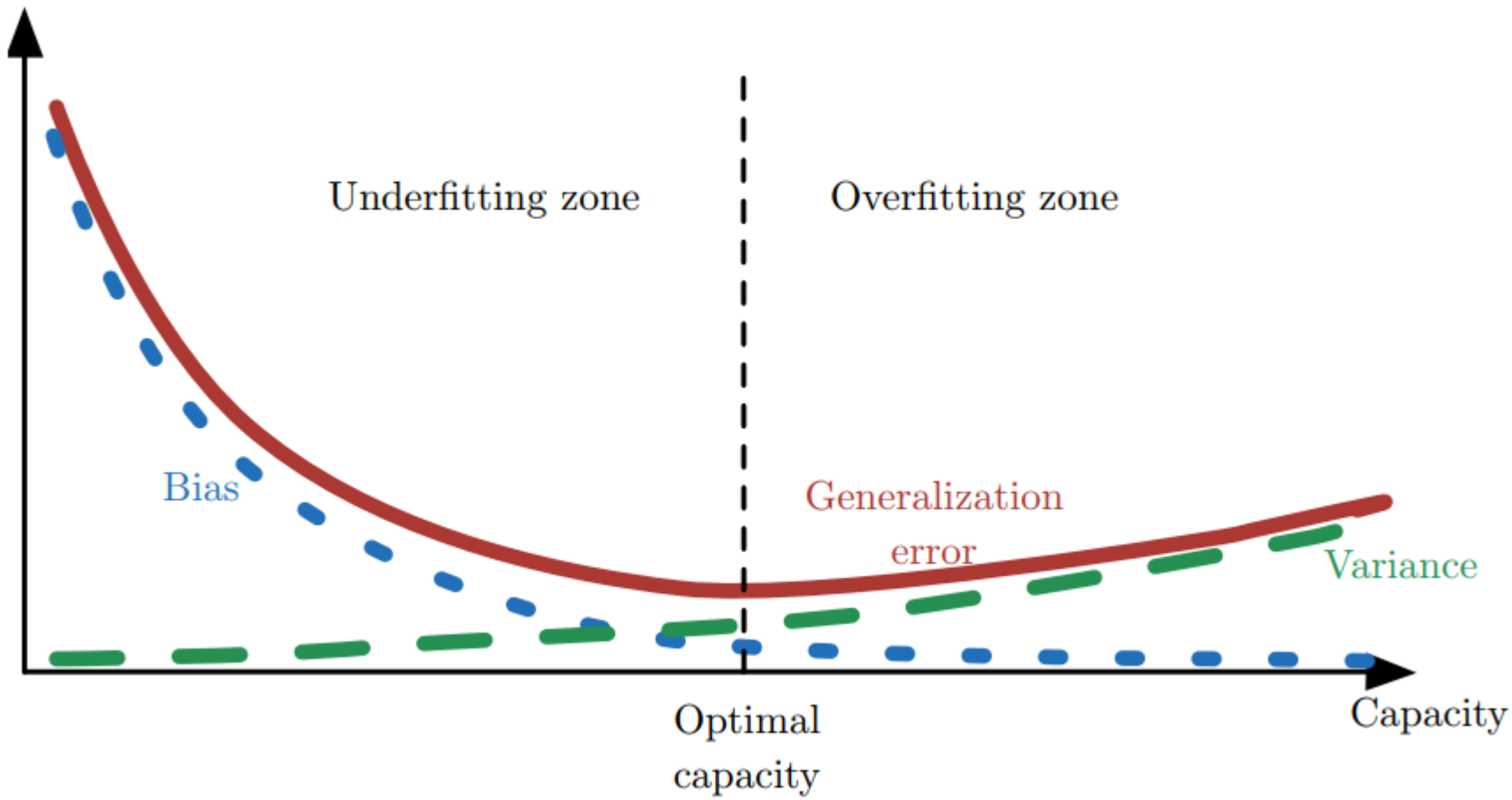
- For an ML problem, we care about the *generalization error* of our algorithm/learned function
  - Generalization error is the error we would obtain if we applied our learned function to new data
  - Test error is an *estimate* of the generalization error
- The generalization error can be formulated as a sum of variance, bias, and noise
- Thus to minimize generalization error, we need a method with
  - Low variance (variability in the learned function if we estimated it using different training data)
  - Low squared bias (error due to estimating the true function using the learned function)

Bias-variance tradeoff

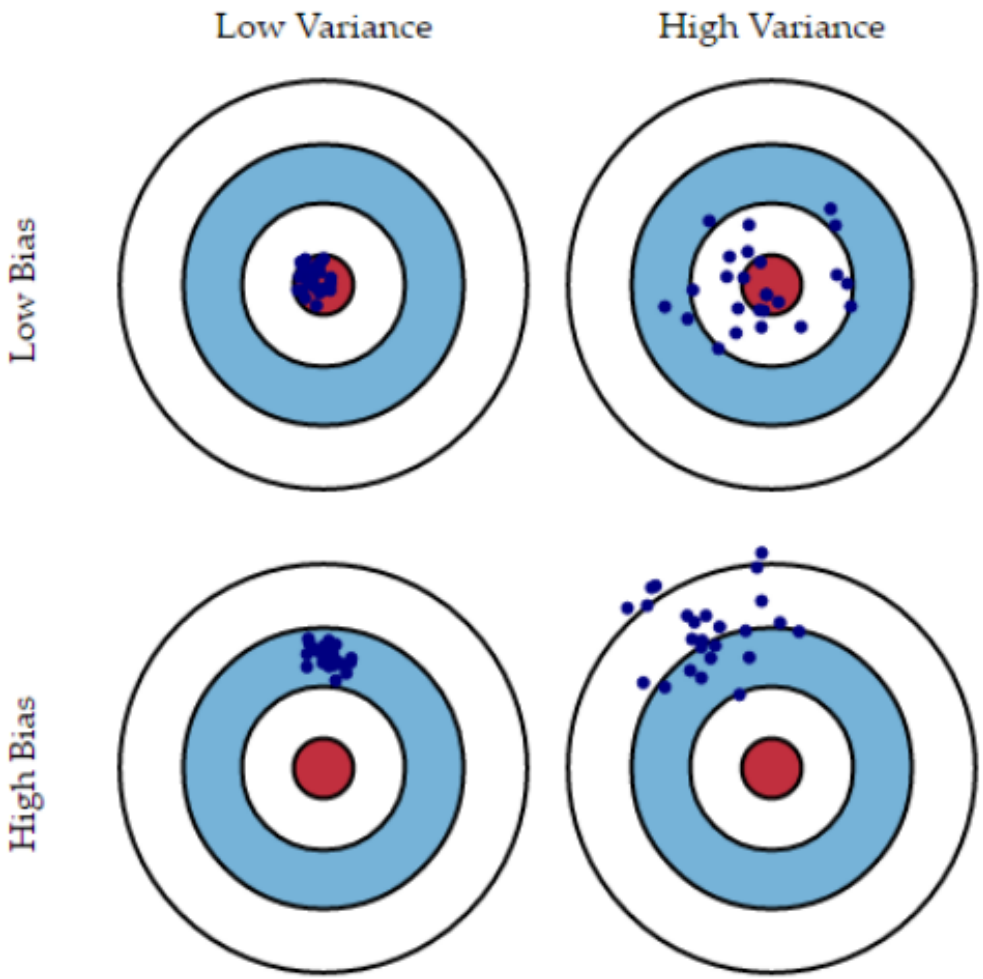


- Typically, variance comes from differences between training sets, bias from model assumptions, and noise from other factors
- Simple methods (e.g. linear methods) tend to have low variance and high bias
  - The learned function is often very different from the true function
- Flexible/complex methods tend to have high variance and low bias
  - Small changes in the data tend to give big changes in the learned function

Bias-variance tradeoff



Bias-variance tradeoff



Taken from http://scott.fortmann-roe.com/docs/BiasVariance.html

Random Forests

Analysis of bagging + decision trees

- Each tree in bagging is identically distributed (i.d.)
- The expectation of an average of B trees is the same as the expectation of any one of them
  - $\mathbb{E}\left[\frac{1}{B}\sum_{i=1}^{B} X_i\right] = \frac{1}{B}\sum_{i=1}^{B} \mathbb{E}[X_i] = \mathbb{E}[X_i]$ if the X_i are i.d. (independence not required)
  - Thus the bias of bagged trees is the same as that of each tree
- Thus we can only improve performance with bagging by reducing variance

Random Forests

Analysis of bagging + decision trees

- Each tree in bagging is identically distributed (i.d.)
- The variance of an average of i.i.d. random variables (each with variance σ^2) is $\frac{1}{B}\sigma^2$
- In contrast, if the variables are only i.d. with positive pairwise correlation ρ between them, then the variance is

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

  - As B increases, the second term disappears but the first remains
  - Thus the correlation between bagged trees limits the benefits of averaging

Random Forests

- **Better idea**: introduce random features
- Each <u>node</u> is based on a randomly selected subset of features
  - Can combine this idea with bagging
  - Not the same as selecting random features for the whole tree
- The random features leads to less correlated predictions
  - Leads to decreased variance of the ensemble prediction
- Rule of thumb (for classification) for the number of randomly selected features per decision tree is $\sqrt{d}$, where d is the total # of features
- Random forests are probably the best "off-the-shelf" method for classification

Johnson-Lindenstrauss lemma

- Provides a viewpoint of why random forests work so well
- The lemma states that a set of points in high dimensions can be embedded into low dimensions while preserving pairwise distances
  - One proof provides a construction using random projections and the concentration of measure
- Given $0 < \epsilon < 1$, a set X of m points in $\mathbb{R}^N$, and a number $n > \frac{8\ln(m)}{\epsilon^2}$, there exists a linear map $f: \mathbb{R}^N \to \mathbb{R}^n$ s.t. $\forall u, v \in X$,

$$(1-\epsilon)\|u-v\|^2 \leq \|f(u)-f(v)\|^2 \leq (1+\epsilon)\|u-v\|^2.$$

- A distributional version exists

Johnson-Lindenstrauss lemma

- How does this connect to random forests?

- In some sense, choosing random features and growing a decision tree from those features that maps from high dimensions to low dimensions could be viewed as a random function
  - The Johnson-Lindenstrauss lemma then indicates that this projection approximates the relationships between inputs quite well

- We'll see more of random forests later

Boosting

Boosting

- Boosting is an ensemble method with *weighted* majority vote
- Assume binary classification: $y \in \{-1,1\}$
- Final classifier has the form

$$h_T(\boldsymbol{x}) = \text{sign}\left\{\sum_{t=1}^{T} \alpha_t f_t(\boldsymbol{x})\right\}$$

  - $f_1, \dots, f_T$ are the *base classifiers*
  - $\alpha_1, \dots, \alpha_T > 0$ reflect the confidence in each base classifier

Boosting

Examples of base classifiers/learners

- Decision trees
- Decision stumps (decision tree with depth 1)
- Radial basis functions
  - $f_i(\boldsymbol{x}) = \text{sign}\{k_\sigma(\boldsymbol{x}, \boldsymbol{x}_i) + b\}$ where k_σ is a radial kernel (e.g., Gaussian)

- Note that the latter two suggest the base classifiers can be very simple

The Boosting Principle

- Basic idea: learn $f_1, \dots, f_T$ sequentially where f_T is produced by the base learner given a weight vector $\boldsymbol{w}^t = (w_1^t, \dots, w_n^t)$ as input
- Weights are updated to place more emphasis on training examples that are harder to classify
- Conceptually $\boldsymbol{w}^t \to \boldsymbol{w}^{t+1}$:
  - If $f_t(\boldsymbol{x}_i) = y_i, w_i^{t+1} < w_i^t$ (downweight)
  - If $f_t(\boldsymbol{x}_i) \neq y_i, w_i^{t+1} > w_i^t$ (upweight)
- The empirical risk that f_{t+1} minimizes is then weighted by w^{t+1}
- Thus f_{t+1} tries harder on samples that f_t misclassified

Adaboost (adaptive boosting)

- First successful boosting algorithm

Algorithm Adaboost

Input: $\{(x_i, y_i)\}_{i=1}^n, T, \mathcal{F}$, base learner

Initialize: $\mathbf{w}' = (\frac{1}{n}, ..., \frac{1}{n})$

for $t = 1, ..., T$ **do**

$\mathbf{w}^t \rightarrow \boxed{\text{base learner}} \rightarrow f_t$

$r_t := \sum_{i=1}^n w_i^t \mathbf{1}_{\{f_t(x_i) \neq y_i\}} = e_{\mathbf{w}^t}(f_t)$

$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - r_t}{r_t} \right)$

$w_i^{t+1} = \frac{w_i^t \exp(-\alpha_t y_i f_t(x_i))}{z_t}$ ← Normalization constant

end for

Output: $h_T(x) = \text{sign} \left\{ \sum_{t=1}^T \alpha_t f_t(x) \right\}$

Weak Learning

- Adaboost is justified by the following:
- Denote $\gamma_t = \frac{1}{2} - r_t$
  - Recall that $r_t = \sum_{i=1}^{n} w_i^t 1_{\{f_t(\boldsymbol{x}_i) \neq y_i\}} = e_{\boldsymbol{w}^t}(f_t)$
    - The weighted error of f_t
- Note that for any f and $\boldsymbol{w}$,

$$e_{\boldsymbol{w}}(f) + e_{\boldsymbol{w}}(-f) = 1$$

- We can assume that $\gamma_t \geq 0$ iff $r_t \leq \frac{1}{2}$
- Otherwise, we could simply replace f_t with $-f_t$

Weak Learning

Theorem: The training error of Adaboost satisfies

$$\frac{1}{n}\sum_{i=1}^{n} 1_{\{h_T(x_i)\neq y_i\}} \leq \exp\left(-2\sum_{t=1}^{T}\gamma_t^2\right).$$

In particular, if $\gamma_t \geq \gamma > 0$ for all t, then

$$\frac{1}{n}\sum_{i=1}^{n} 1_{\{h_T(x_i)\neq y_i\}} \leq \exp(-2T\gamma^2)$$

- $r_t = \frac{1}{2} \Rightarrow f_t$ is randomly guessing
- Thus $\gamma_t \geq \gamma > 0$ means f_t is at least slightly better than random guessing

Weak Learning

- If the base learner is guaranteed to satisfy $\gamma_t \geq \gamma > 0$ for all t, then it is said to satisfy the <u>weak learning hypothesis</u>
- The theorem says that under the weak learning hypothesis, the Adaboost training error converges to zero exponentially fast
- To avoid overfitting, the parameter T should be chosen carefully
  - E.g. by cross-validation

Boosting as Functional Gradient Descent

- Adaboost can be viewed as an iterative algorithm for minimizing the empirical risk corresponding to the exponential loss
- Generalizing the loss gives us different boosting algorithms with different properties
  - Referred to as <u>gradient boosting</u> techniques

Boosting as Functional Gradient Descent

- For a fixed base learner class $\mathcal{F}$, define

$$\mathrm{span}(\mathcal{F}) = \left\{ \sum_{t=1}^{T} \alpha_t f_t \,\middle|\, T \geq 1, \alpha_t \in \mathbb{R}, f_t \in \mathcal{F} \right\}$$

- Now consider the problem:

$$\min_{F \in \mathrm{span}(\mathcal{F})} \frac{1}{n} \sum_{i=1}^{n} 1_{\{\mathrm{sign}(F(\boldsymbol{x}_i)) \neq y_i\}}$$

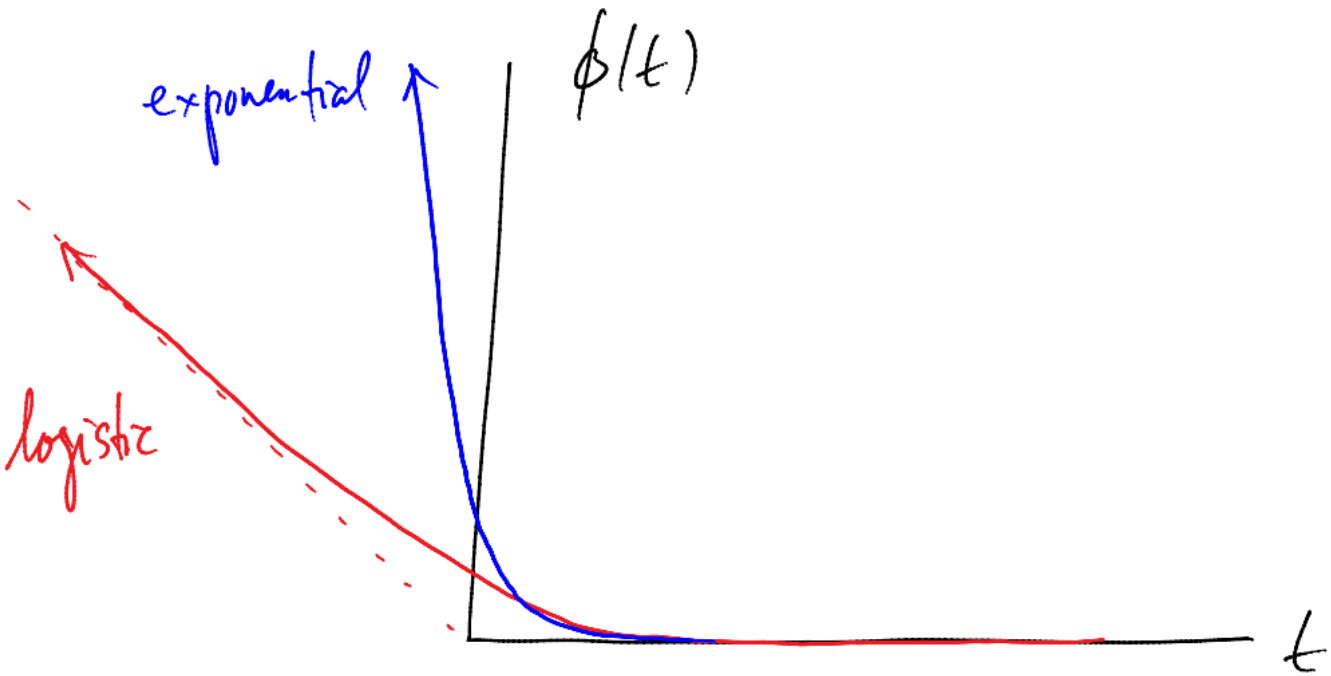
- As before, we replace the 0-1 loss with a surrogate loss ϕ (for computational reasons):

$$\min_{F \in \mathrm{span}(\mathcal{F})} \frac{1}{n} \sum_{i=1}^{n} \phi\big(y_i F(\boldsymbol{x}_i)\big)$$

Boosting as Functional Gradient Descent

Examples of surrogate losses:

- $\phi(t) = \exp(-t)$ (exponential loss)
- $\phi(t) = \log(1 + \exp(-t))$ (logistic loss)
- $\phi(t) = \max(0,1 - t)$ (hinge loss)



Boosting as Functional Gradient Descent

- Assume ϕ is differentiable and $\phi' < 0$ everywhere
- We can solve the optimization problem using <u>functional gradient descent</u> (FGD)
  - Gradient descent on a space consisting of functions
- At the tth iteration of FGD, the current iterate is

$$F_{t-1} = \sum_{s=1}^{t-1} \alpha_s f_s \,.$$

- The next iterate has the form

$$F_t = F_{t-1} + \alpha_t f_t$$

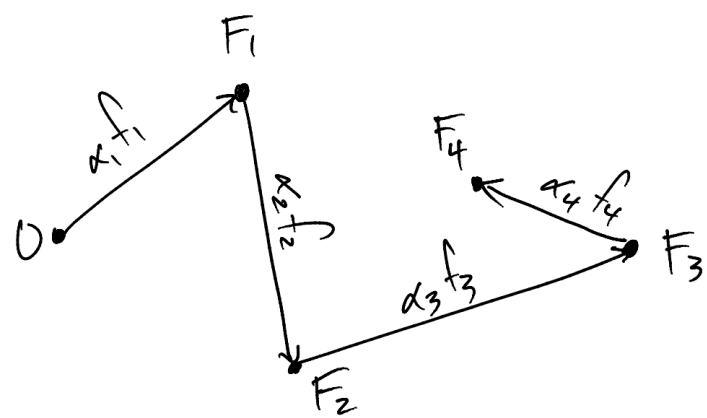
Boosting as Functional Gradient Descent

- View $\alpha_1, f_1, \ldots, \alpha_{t-1}, f_{t-1}$ as fixed and define

$$B_t(\alpha, f) = \frac{1}{n}\sum_{i=1}^{n} \phi\big(y_i F_{t-1}(\boldsymbol{x}_i) + y_i \alpha f(\boldsymbol{x}_i)\big)$$

- The α_t, f_t are chosen by

1. f_t = function $f \in \mathcal{F}$ for which the directional derivative of B_t in the direction of f is minimized
2. α_t = stepsize $\alpha > 0$ in the direction f_t for which $B_t(\alpha, f_t)$ is minimized



Details (1)

$$\left.\frac{\partial B(\alpha, f)}{\partial \alpha}\right|_{\alpha=0} = \frac{1}{n}\sum_{i=1}^{n} y_i f(\boldsymbol{x}_i)\phi'\big(y_i F_{t-1}(\boldsymbol{x}_i)\big)$$

- Minimizing this wrt f is equivalent to minimizing:

$$-\sum_{i=1}^{n} y_i f(\boldsymbol{x}_i) \cdot \boxed{\frac{\phi'\big(y_i F_{t-1}(\boldsymbol{x}_i)\big)}{\sum_{j=1}^{n} \phi'\Big(y_j F_{t-1}\big(\boldsymbol{x}_j\big)\Big)}}$$

Since $\phi' < 0$

Call this w_i^t

$$= \sum_{i=1}^{n} w_i^t 1_{\{f(\boldsymbol{x}_i) \neq y_i\}} - \sum_{i=1}^{n} w_i^t 1_{\{f(\boldsymbol{x}_i) = y_i\}}$$

$$= 2\left(\sum_{i=1}^{n} w_i^t 1_{\{f(\boldsymbol{x}_i) \neq y_i\}}\right) - 1$$

Solve this by just applying the base learner

Details (2)

$$\alpha_t = \arg\min_{\alpha} B_t(\alpha, f_t)$$
$$= \arg\min_{\alpha} \frac{1}{n} \sum_{i=1}^{n} \phi\big(y_i F_{t-1}(\boldsymbol{x}_i) + y_i \alpha f_t(\boldsymbol{x}_i)\big)$$

- This is just a scalar minimization problem
  - Solve either numerically (e.g. gradient descent) or analytically if possible

Gradient Boosting Summary

- **Input**: $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n, T, \mathcal{F}$, base learner, surrogate loss ϕ (differentiable, $\phi' < 0$ everywhere)
- **Initialize**: $\boldsymbol{w}^1 = \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$, $F_0 = 0$
- **For** $t = 1, \dots, T$
  - $\mathbf{w}^t \rightarrow \boxed{\text{base learner}} \rightarrow f_t$
  - $\alpha_t = \arg\min_{\alpha} \frac{1}{n} \sum_{i=1}^n \phi\left(y_i F_{t-1}(\boldsymbol{x}_i) + y_i \alpha f_t(\boldsymbol{x}_i)\right)$
  - $F_t = F_{t-1} + \alpha_t f_t$
  - $w_i^{t+1} = \frac{\phi'\left(y_i F_{t-1}(\boldsymbol{x}_i)\right)}{\sum_{j=1}^n \phi'\left(y_j F_{t-1}(\boldsymbol{x}_j)\right)}$
- **End**
- **Output**: $h_T(\boldsymbol{x}) = \text{sign}\left(F_T(\boldsymbol{x})\right)$

Connection to Adaboost

- $\phi(t) = e^{-t} \Rightarrow$ algorithm specializes to Adaboost
- Advantages of exponential loss:
  - w_i^t has a multiplicative update stemming from the property that $\phi'(a+b) = -\phi'(a)\phi'(b)$
  - α_t has a closed form expression
- For other losses, the updates are less simple but still not difficult to implement
- Other losses may have better statistical properties
  - E.g., logistic loss is much less sensitive to outliers than exponential loss

XGBoost

XGBoost

- A regularized (mostly) tree-based boosting method
- Popular now due to its success on Kaggle
- Uses 2nd order optimization (e.g. Newton's method)
- Efficient implementation, especially on distributed systems
- Automated feature selection

XGBoost

- With training data $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, add one tree at a time
- Define $F_t(\boldsymbol{x}_i) = F_{t-1}(\boldsymbol{x}_i) + f_t(\boldsymbol{x}_i)$
  - $F_0(\boldsymbol{x}_i) = 0$
- Total loss function at step t:

$$obj_t = \sum_{i=1}^{n} L\big(y_i, F_t(\boldsymbol{x}_i)\big) + \omega(f_t)$$

$$= \sum_{i=1}^{n} L\big(y_i, F_{t-1}(\boldsymbol{x}_i) + f_t(\boldsymbol{x}_i)\big) + \omega(f_t)$$

  - L is our loss function, ω is a regularizer

XGBoost

- Total loss function at step t:

$$\sum_{i=1}^{n} L\big(y_i, F_{t-1}(\boldsymbol{x}_i) + f_t(\boldsymbol{x}_i)\big) + \omega(f_t)$$

- Do a Taylor approximation:

$$obj_t = \sum_{i=1}^{n} \left[L\big(y_i, F_{t-1}(\boldsymbol{x}_i)\big) + g_i f_t(\boldsymbol{x}_i) + \frac{1}{2} h_i f_t^2(\boldsymbol{x}_i) \right] + \omega(f_t)$$

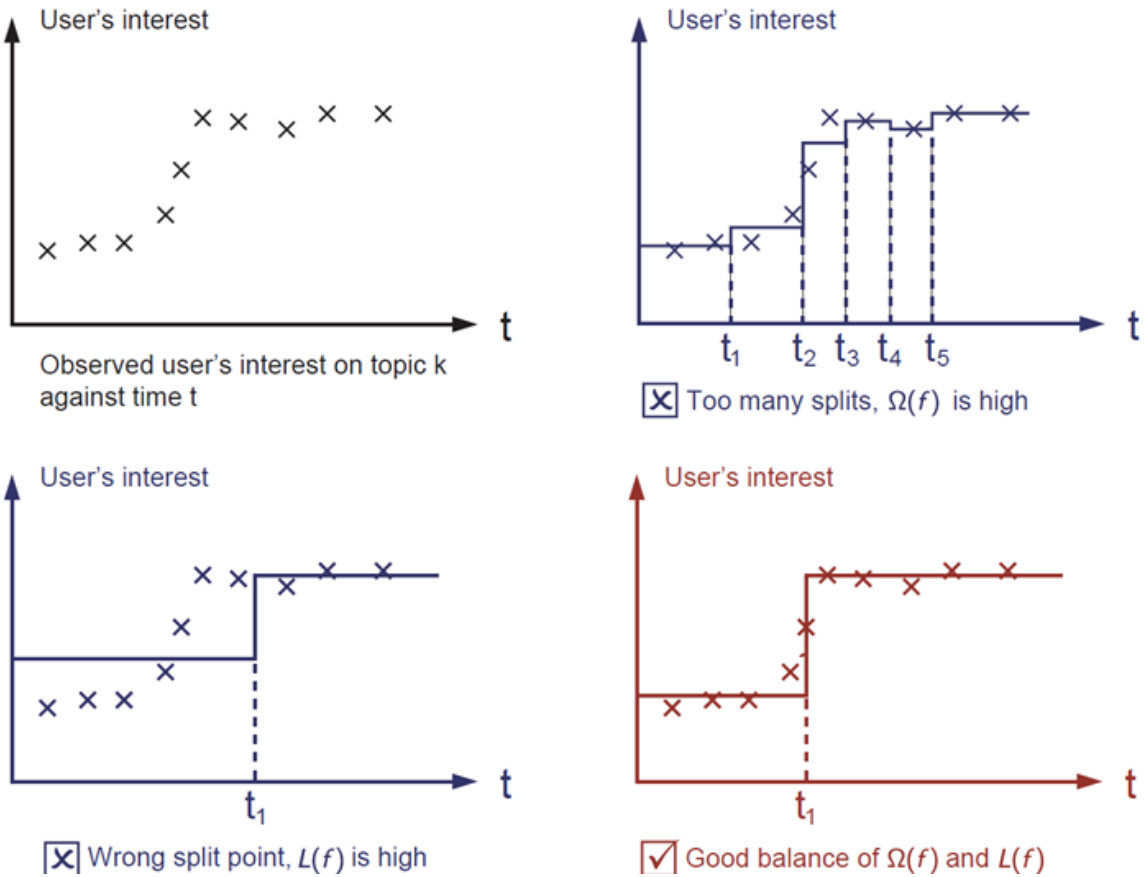
  - g_i and h_i are the gradient and Hessian, respectively:

$$g_i = \left[\frac{\partial L\big(y_i, f(\boldsymbol{x}_i)\big)}{\partial f(\boldsymbol{x}_i)} \right]_{f(\boldsymbol{x}_i)=F_{t-1}(\boldsymbol{x}_i)}, h_i = \left[\frac{\partial^2 L\big(y_i, f(\boldsymbol{x}_i)\big)}{\partial f(\boldsymbol{x}_i)^2} \right]_{f(\boldsymbol{x}_i)=F_{t-1}(\boldsymbol{x}_i)}$$

- First term $L\big(y_i, F_{t-1}(\boldsymbol{x}_i)\big)$ is a constant and can be dropped

XGBoost Regularization

- Regularization helps prevent overfitting



XGBoost Regularizer

- How do we regularize?
- Many options, but the following works well in practice:
- A decision tree can be defined as $f_t(\boldsymbol{x}) = w_{q(\boldsymbol{x})}$
  - $\boldsymbol{w} \in \mathbb{R}^M$ is the vector of scores (i.e. output) on leaves
  - $q: \mathbb{R}^d \to \{1,2,\dots,M\}$ assigns each data point to its corresponding leaf
  - M = # of leaves in the tree
- The complexity in XGBoost is defined as

$$\omega(f) = \gamma M + \frac{1}{2}\lambda \sum_{j=1}^{M} w_j^2$$

Decision tree scores

- A decision tree can be defined as $f_t(\boldsymbol{x}) = w_{q(\boldsymbol{x})}$
- In regression, w_i outputs a real number
- In classification, w_i outputs either a class label OR a real number
  - The latter case is similar to logistic regression, giving some notion of confidence
- XGBoost computes $\boldsymbol{w}$ using its objective function

XGBoost Scores

- Updated objective function:

$$obj_t = \sum_{i=1}^{n} \left[g_i w_{q(\boldsymbol{x}_i)} + \frac{1}{2} h_i w_{q(\boldsymbol{x}_i)}^2 \right] + \gamma M + \frac{1}{2} \lambda \sum_{j=1}^{M} w_j^2$$

$$= \sum_{j=1}^{M} \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma M$$

  - $I_j = \{ i | q(\boldsymbol{x}_i) = j \}$ is the set of indices assigned to the j-th leaf

- Compress further with $G_j = \sum_{i \in I_j} g_i$ and $H_j = \sum_{i \in I_j} h_i$

$$obj_t = \sum_{j=1}^{M} \left[G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma M$$

XGBoost Scores

$$obj_t = \sum_{j=1}^{M} \left[G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2 \right] + \gamma M$$

- Optimizing this equation for w_j gives

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

$$obj_t^* = -\frac{1}{2} \sum_{j=1}^{M} \frac{G_j^2}{H_j + \lambda} + \gamma M$$

- The final score is like an impurity measure of the decision tree that also considers model complexity

Learning the tree structure

- Ideally, we could consider all possible trees and choose the tree with the best value of obj_t^*
- But this is intractable, so we use a greedy approach
- Consider the "gain" at an attempted split:

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$$

Left leaf score Right leaf score Parent score Regularization

- Choose the split with the largest gain
- How can this be parallelized?

XGBoost Final Comments

- Can adapt to different loss functions and weak learners
- Uses data summaries (e.g. quantiles) to speed up training
- Can also subsample features like random forests
- Has implementations in many languages including distributed environments
- Recommended if you are working with tabular data and need good results
- CatBoost is an alternative approach that is also very popular

Big Picture

- Simple classifiers often perform poorly on data and/or can be unstable
  - Histogram classifiers
  - Decision trees
- Taking a (weighted) majority vote from an ensemble of these simple classifiers can lead to a superior classifier that is stable
- This idea also extends to regression (and estimation as we'll see later)

Further reading

- ESL Chapters 10, 15, and 16, Sections 2.9, 7.2, and 7.3
- ISL Sections 2.2 and 8.2
- XGBoost: https://xgboost.readthedocs.io/en/stable/index.html