Boosting 2

-Introduction to Data Science-

Lecturer: Darren Homrighausen, PhD

Preamble:

- Nonparametric methods provide a flexible fit but can only be used in low dimensions
- Boosting is an algorithm for fitting a greedy, stepwise nonparametric procedure
- Unlike nonparametric methods, the basis ϕ is also estimated from the data
- Regularization via a learning rate λ and the number of steps B is important

Boosting so far...

Summary: In the previous lecture, we made the following observations

- When p is very small, we can flexibly fit a local average to estimate the Bayes' rule

  (e.g. splines)
- When p is larger, we need to limit the flexibility

  (e.g. with generalized additive models (GAMs) or GLMs ($\beta^\top X$))
- One drawback is that GAMs and GLMs encode strong assumptions

  (In particular, no interactions between features)

How does boosting work?

Boosting fits a type of nonparametric model

$$f(X) = \sum_{b=1}^{B} \beta_b \phi_b(X) = \beta^\top \Phi(X)$$

where

- β are weights
- ϕ is some base learner

The base learner will be a fixed family of procedures that depend on some parameters, θ

So, we will write $\phi_b(X) = \phi(X, \theta_b)$

Example: ϕ could be a tree. Then θ_b would be the split points, the values for the decisions made at each split point, and the predictions at each terminal node

How does boosting work?

Nonparametrics:

$$f(X) = \sum_{j=1}^{K} \beta_j \phi_j(X)$$

Which we can fit via

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \left|\left| Y - \Phi\beta \right|\right|_2^2 ,$$

and form $\hat{f}(X) = \hat{\beta}^\top \Phi(X)$. Note:

- all the coefficients β are estimated at the same time
- the basis ϕ_k is specified before hand (e.g. splines)

What if instead we estimate both the coefficients and basis?

This creates a nonconvex optimization problem

$\rightarrow$ fit in a greedy, stepwise manner

Forward stepwise nonparametrics

Specify a starting point $\hat{f}(X) = 0$

For $b = 1, \ldots, B$

1. Fit: $\hat{\beta}_b, \hat{\theta}_b = \text{argmin}_{\beta,\theta} \sum_{i=1}^n \ell(\hat{f}(X_i) + \beta\phi(X_i, \theta), Y_i)$
2. Set: $\hat{f}_b(X) = \hat{\beta}_b \phi(X; \hat{\theta}_b)$
3. Update: $\hat{f}(X) \leftarrow \hat{f}(X) + \hat{f}_b(X)$

Under squared error loss $\ell(f(X), Y) = (Y - f(X))^2$

$$\begin{aligned} \ell(\hat{f}(X_i) + \beta\phi(X_i, \theta), Y_i) &= (Y_i - \hat{f}(X_i) - \beta\phi(X_i, \theta))^2 \\ &= (\tilde{R}_i - \beta\phi(X_i, \theta))^2 \end{aligned}$$

where $\tilde{R}_i$ is the i^{th} residual from $\hat{f}$

Hence, finding the $\hat{f}_b$ means fitting to the residuals...

Back to Boosting for Regression

Reminder: Boosting regression trees

Set $\hat{f} \equiv 0$ and $R = Y \in \mathbb{R}^n$. For $b = 1, \ldots, B$, do:

Forward, stepwise nonparametric regression:

1. Fit: $\hat{\beta}_b, \hat{\theta}_b = \text{argmin}_{\beta,\theta} \sum_{i=1}^n (\tilde{R}_i - \beta\phi(X_i, \theta))^2$
2. Set: $\hat{f}_b(X) = \hat{\beta}_b \phi(X; \hat{\theta}_b)$
3. Update: $\hat{f}(X) \leftarrow \hat{f}(X) + \hat{f}_b(X)$

Boosting for regression:

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

These are the same, except for two minor differences:

- Boosting includes the learning rate λ
- A slight difference in how R and $\tilde{R}$ are defined

(These differences are to reduce overfitting. See ESL 10.10 and/or "boostingExtras.pdf" for extra details)

Boosting for classification

Forward stepwise nonparametrics

As boosting can be seen as a greedy, stepwise fit:

$$\hat{\beta}_b, \hat{\theta}_b = \underset{\beta,\theta}{\text{argmin}} \sum_{i=1}^{n} \ell(\hat{f}(X_i) + \beta\phi(X_i, \theta), Y_i)$$

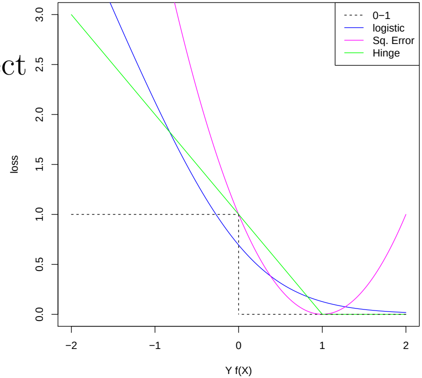
If we change the loss function, we get a different boosting procedure

We'll want to choose a new loss for classification as squared error loss doesn't work well

Loss functions for classification

$Y \in \{-1, 1\}$

$Yf(X)$ is $\begin{cases} > 0 & \text{if correct} \\ < 0 & \text{if incorrect} \end{cases}$



$f(X)$ is (signed) distance from decision boundary

$\rightarrow$ loss functions for classification should seek to make $Yf(X)$ as large as possible

Boosting for classification

We can adapt boosting to classification by changing the loss function

This will produce an additive model $\hat{f}(X) = \sum_{b=1}^{B} \lambda \hat{f}_b(X)$
(Again, $\hat{f}_b$ is the trained base learner that minimizes the training error at the b^{th} step)

This gets converted to a classifier via $\hat{g}(X) = \mathrm{sgn}(\hat{f}(X))$

For classification, two common choices of ℓ are:

- Adaboost
- Bernoulli or Logistic

Boosting for classification

More details:

- Adaboost: Iteratively fits a classifier on reweighted training data such that misclassified observations are upweighted

  It turns out this is equivalent to stepwise nonparametrics with the exponential loss function:

$$\ell(f(X), Y) = \exp\{-f(X)Y\}$$

- Bernoulli or Logistic: If we assume a Bernoulli distribution with logistic loss, we acquire another stepwise nonparametric procedure:

$$\ell(f(X), Y) = \log\left(1 + \exp\{-2Yf(X)\}\right)$$

(Note that the '2' is from us defining $Y \in \{-1, 1\}$ whereas the Bernoulli is commonly written $Y \in \{0, 1\}$ and hence our label is "twice" the usual label)

AdaBoost outline

We give an overview of ‘AdaBoost.M1.’

(Freund and Schapire (1997))

Select a base classifier that you want to boost

(E.g. a tree with a very small number of terminal nodes)

First, train the base classifier as usual on the training data $\mathcal{D}$

Then start iterating $b = 1, 2, \ldots B$,

At each step b,

1. the observations are re-weighted to increase the weights of misclassified observations

   (Implicitly, this lowers the weight on correctly classified observations)

2. A new classifier is trained on the re-weighted training data

(Discrete) AdaBoost algorithm

Assume $Y \in \{-1, 1\}$

1. Initialize $w_i \equiv 1/n$ for $i = 1, \dots, n$
2. For $b = 1, \dots, B$
   2.1 Fit the base classifier on $\mathcal{D}$, weighted by $w_i \to \hat{g}_b$
   2.2 Compute
   $$\hat{R}_b = \frac{\sum_{i=1}^n w_i \mathbf{1}(Y_i \neq \hat{g}_b(X_i))}{\sum_{i=1}^n w_i}$$
   2.3 Find $\hat{\beta}_b = \log((1 - \hat{R}_b)/\hat{R}_b)$
   2.4 Set $w_i \leftarrow w_i \exp\{\hat{\beta}_b \mathbf{1}(Y_i \neq \hat{g}_b(X_i))\}$
3. Output: $\hat{g}(X) = \operatorname{sgn}\left(\sum_{b=1}^B \hat{\beta}_b \hat{g}_b(X)\right)$

Some supporting simulations

Simulation: Equal probability

Base learner: 'depth 2-stumps'

(These are trees, but constrained to have no more than 4 terminal nodes)

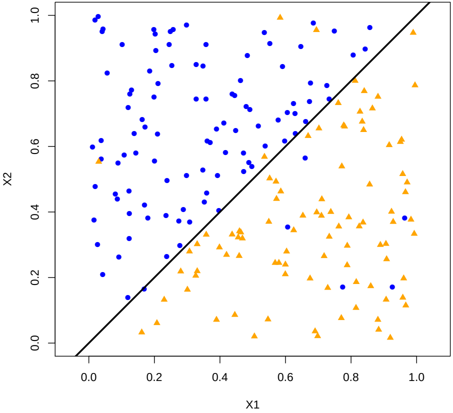
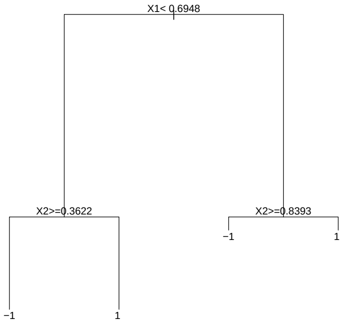
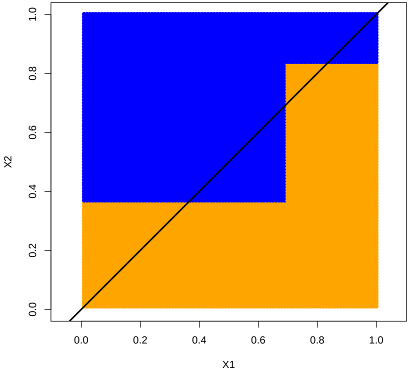


Figure: The solid, black line is the Bayes' rule w.r.t. 0 -1 loss

Simulation: Equal probability

Using a depth-2 stump:

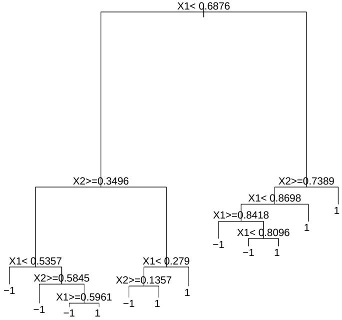


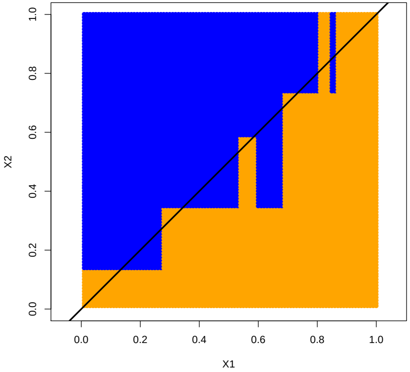


Simulation: Equal probability

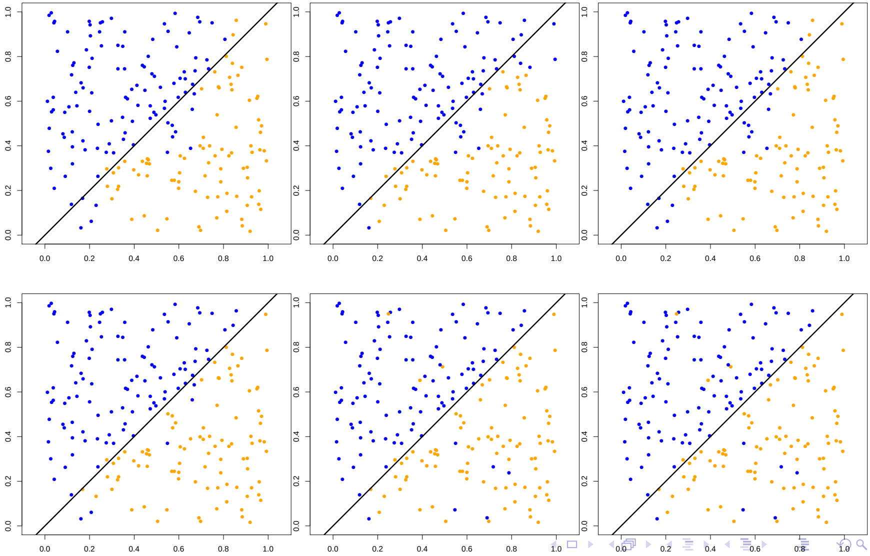
Using an unpruned tree:

(Note that I used rpart, which parameterizes splits differently than tree)

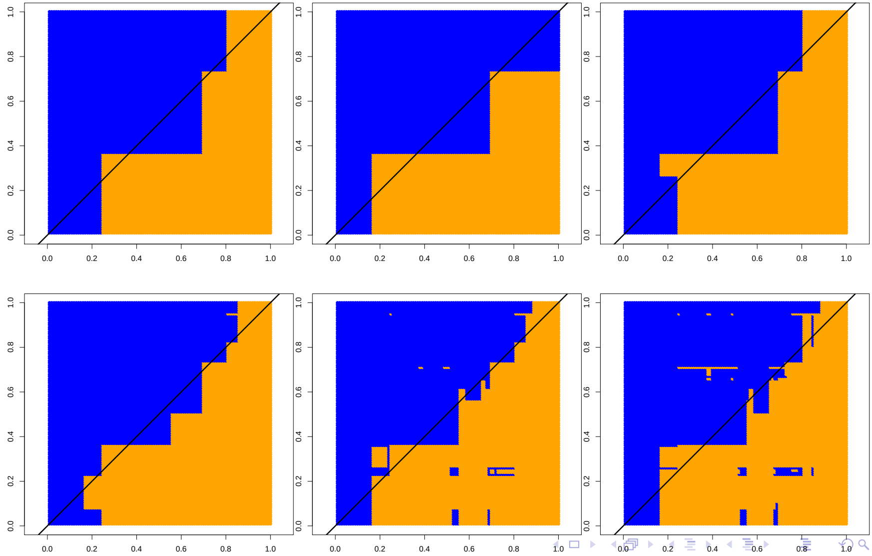




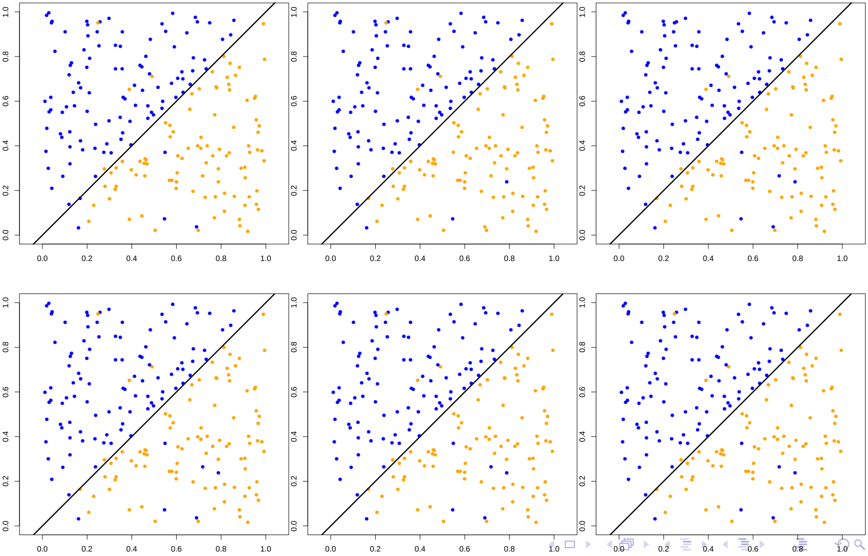
AdaBoost: Increasing B (train)



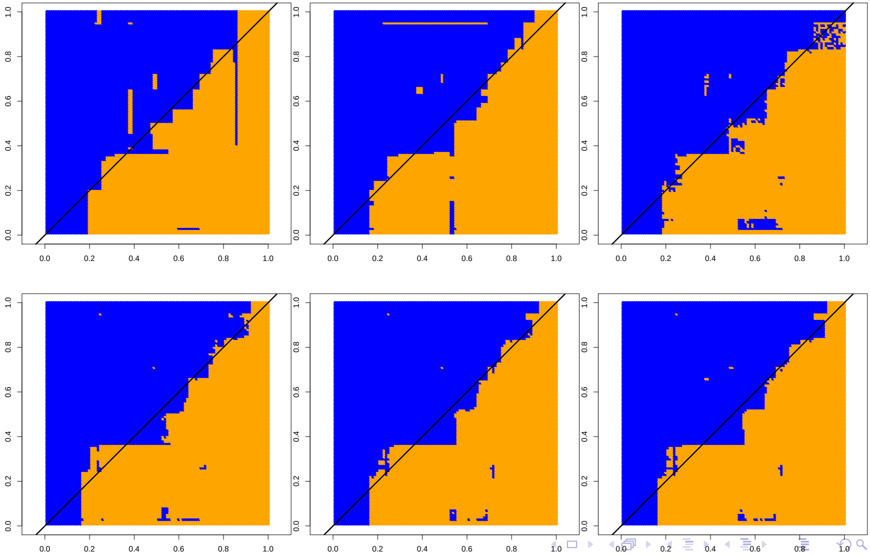
AdaBoost: Increasing B (test)



Random Forest: Increasing B (train)



Random Forest: Increasing B (test)



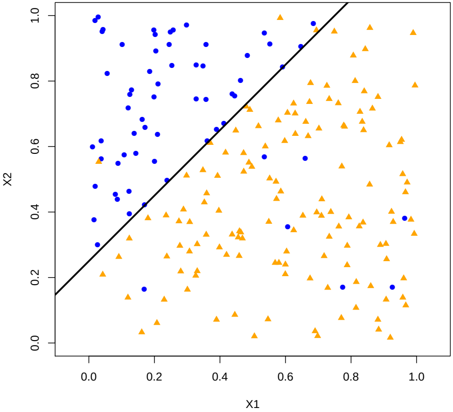
Results: Confusion matrices

(These are at best B solution)

			Truth		
			-1	1	Mis-Class
Our Preds	Unpruned	-1	84	18	
		1	11	87	14.5%
	Stump	-1	77	16	
		1	18	89	17%
	Boost	-1	92	8	
		1	5	92	8%
	RF	-1	87	9	
		1	8	96	8.5%

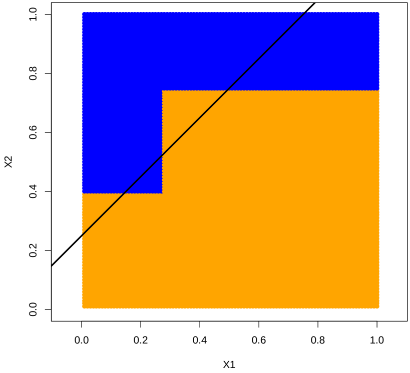
Simulation: Unequal probability

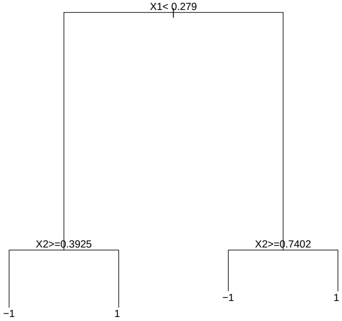
Let's change the simulation so that the class probabilities aren't the same



Simulation: Unequal probability

Using a depth-2 stump:

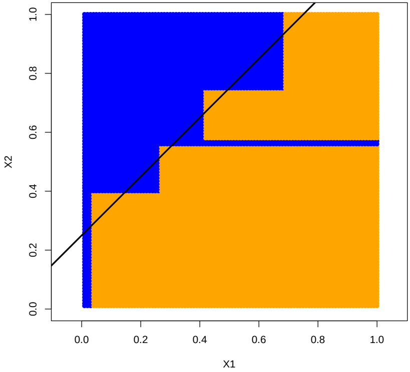


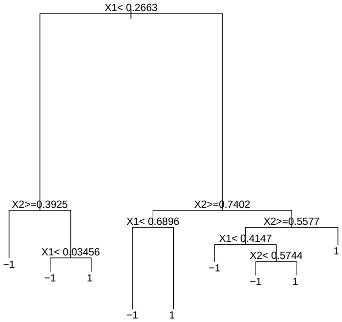


Simulation: Unequal probability

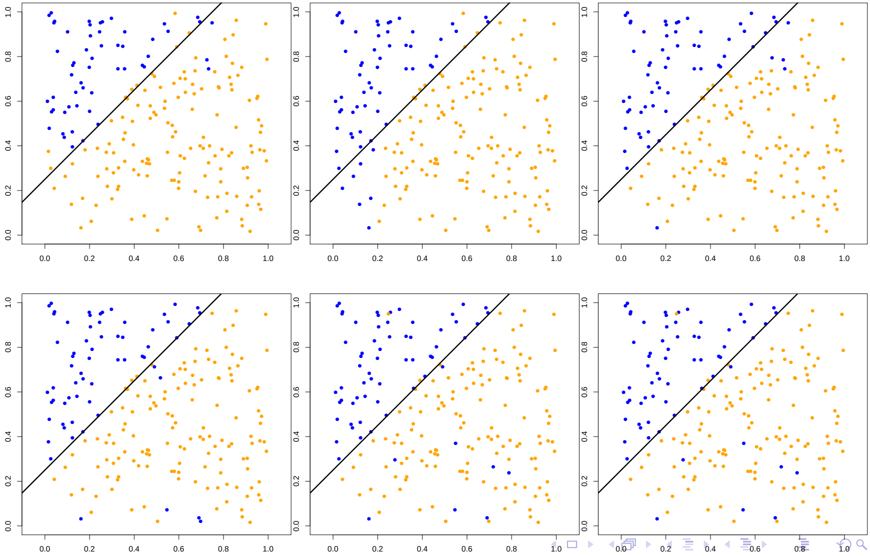
Using an unpruned tree:

(Note that I used rpart, which parameterizes splits differently than tree)

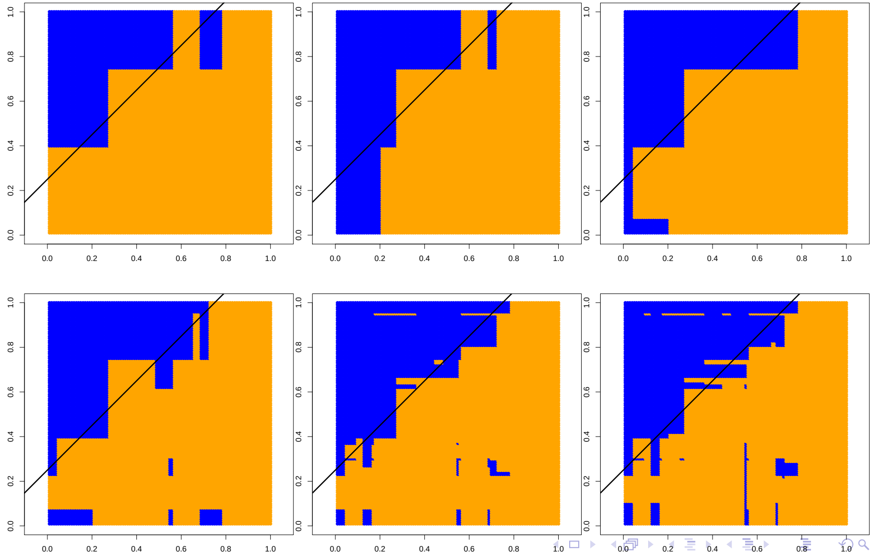




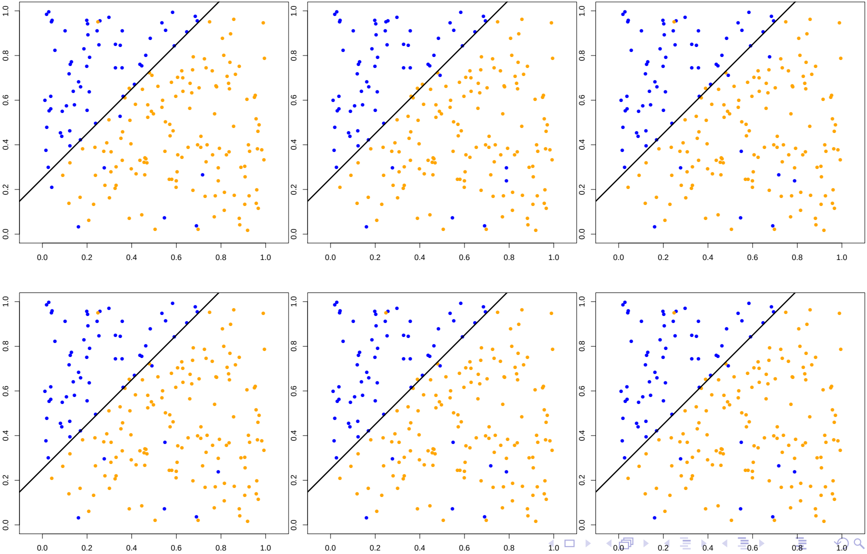
AdaBoost: Increasing B (train)



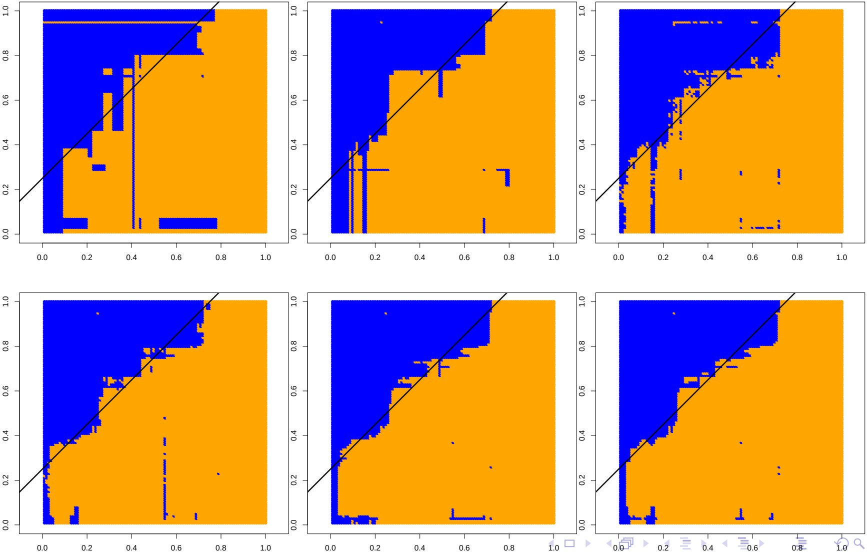
AdaBoost: Increasing B (test)



Random Forest: Increasing B (train)



Random Forest: Increasing B (test)



Results: Confusion matrices

(These are at best B solution)

			Truth		
			-1	1	Mis-Class
Our Preds	Unpruned	-1	51	10	
		1	11	128	10.5%
	Stump	-1	50	22	
		1	12	116	17%
	Boost	-1	51	12	
		1	3	134	7.5%
	RF	-1	53	5	
		1	9	133	7%

Discrete AdaBoost

This algorithm became known as 'discrete AdaBoost'

(This is due to the base classifier returning a discrete label)

Output: $\hat{g}(X) = \mathrm{sgn}\left(\sum_{b=1}^{B} \hat{\beta}_b \hat{g}_b(X)\right)$

This was adapted to real-valued predictions in Real AdaBoost

(In particular, probability estimates)

Real AdaBoost

Assume $Y \in \{-1, 1\}$

1. Initialize $w_i \equiv 1/n$ for $i = 1, \ldots, n$
2. For $b = 1, \ldots, B$
   2.1 Fit the base classifier on $\mathcal{D}$, weighted by w_i $\to \hat{p}_b(X) = \hat{\mathbb{P}}(Y = 1|X)$
   2.2 Set $\hat{f}_b(X) \leftarrow \frac{1}{2} \log(\hat{p}_b(X)/(1 - \hat{p}_b(X)))$
   2.3 Set $w_i \leftarrow w_i \exp\{-Y_i \hat{f}_b(X_i)\}$
3. Output: $\hat{g}(X) = \operatorname{sgn}\left(\sum_{b=1}^{B} \hat{f}_b(X)\right)$

This is referred to as Real AdaBoost and it used the class probability estimates to construct the contribution of the b^{th} classifier, instead of the estimated label

Real AdaBoost

Assume $Y \in \{-1, 1\}$

1. Initialize $w_i \equiv 1/n$ for $i = 1, \ldots, n$
2. For $b = 1, \ldots, B$
   2.1 Fit the base classifier on $\mathcal{D}$, weighted by w_i
   $\rightarrow \hat{p}_b(X) = \hat{\mathbb{P}}(Y = 1|X)$
   2.2 Set $\hat{f}_b(X) \leftarrow \frac{1}{2}\log(\hat{p}_b(X)/(1 - \hat{p}_b(X)))$
   2.3 Set $w_i \leftarrow w_i \exp\{-Y_i \hat{f}_b(X_i)\}$
3. Output: $\hat{g}(X) = \operatorname{sgn}\left(\sum_{b=1}^{B} \hat{f}_b(X)\right)$

This is referred to as Real AdaBoost and it used the class probability estimates to construct the contribution of the b^{th} classifier, instead of the estimated label

(Care is needed when computing $\hat{f}_b$ in practice due to numerical issues with probabilities near 0 or 1. We need to be sure that induced weights are nonnegative.)

(Discrete) AdaBoost interpretation

Forward stagewise additive modeling:

(Using a general loss ℓ)

1. $\hat{\beta}_b, \hat{\theta}_b = \text{argmin}_{\beta,\theta} \sum_{i=1}^{n} \ell(Y_i, \hat{f}(X_i) + \beta\phi(X_i, \theta))$
2. Set $\hat{f}(X) = \hat{f}(X) + \hat{\beta}_b \phi(X; \hat{\theta}_b)$

(Discrete) AdaBoost implicitly does this via the exponential loss function

$$\ell(Y, f) = \exp\{-Yf(X)\}$$

w/ basis function/base classifier/base learner $\phi(X, \theta) = g_b(X)$

(See 'boostingExtra.pdf' notes for the details on this connection)

Why exponential loss?

It can be shown that the Bayes' rule with respect to exponential loss is

$$\operatorname*{argmin}_f \mathbb{E}[\exp\{-Yf(X)\}] = \frac{1}{2}\log\left(\frac{\mathbb{P}(Y=1|X)}{\mathbb{P}(Y=-1|X)}\right)$$

Hence, we are estimating (half) the log odds

$\rightarrow$ use the sgn rule for classification

This is the same form for the Bayes' rule for the logistic loss

$$\ell(f(X), Y) = \log\left(1 + \exp\{-2Yf(X)\}\right)$$

(They give similar results, though there is some evidence that logistic tends to get lower misclassification rates in practice)

Boosting for classification

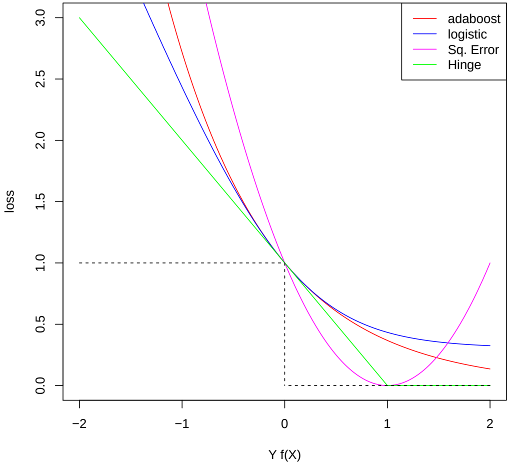


Figure: Here, I've rescaled the logistic loss so that it is easier to compare to Adaboost

Regularization in boosting

Boosting benefits from regularization

(We are minimizing the training error in a stepwise fashion, after all)

In boosting, this is usually done via

- Choosing the parameter B via a risk estimate
- Including a learning rate

$$\hat{f}(X) = \hat{f}(X) + \lambda \hat{f}_b(X)$$

There is a strong interaction between these two parameters

It has been observed repeatedly that

- setting λ to a small constant achieves lower risk than
- not including λ at all

(This is one aspect of the general philosophy of learning slow. If possible, make a huge number of tiny improvement instead of a small number of large improvements)

Next lecture

Discuss two current, popular algorithms and their R implementations

- GBM
- XGBoost

Postamble:

- Nonparametric methods provides a flexible fit, but can only be used in low dimensions
- Boosting is an algorithm for fitting a greedy, stepwise nonparametric procedure

  (Specify a loss function and iteratively minimize the training error)
- Unlike nonparametric methods, the basis ϕ is also estimated from the data

  (In this case, the basis function ϕ is known as the base learner or base classifier)
- Regularization via a learning rate λ and the number of steps B is important

  (Set λ to be a small number and choose B via a risk estimation procedure like K-fold CV)