Classification via Boosting

-Statistical Machine Learning-

Lecturer: Darren Homrighausen, PhD

History

Boosting was proposed in the computational learning literature in a series of papers:

- Schapire (1990)
- Freund (1995)
- Freund and Schapire (1997)

Let's examine a bit the history of boosting, through these three papers

PAC-learning for classification

The first simple boosting procedure was developed using the PAC-learning framework

We covered PAC-learning in the case of generalization error bounds for lasso

In classification, there is more of a computer science flair to notation/terminology that is helpful to know when reading the literature

PAC-learning for classification

Let's introduce the following notation[1]

- A Concept c is a Boolean function on some domain of Instances $\mathcal{X}$ and contained in a Concept class $\mathcal{C}$

  (Here think of concept $\leftrightarrow$ prediction procedure and $\mathcal{X}$ as the domain of the covariates. Also, the concept class $\leftrightarrow$ parameter space)

- The learner is assumed to have access to a source of examples EX, that is randomly and independently drawn from $\mathcal{X}$ according to a fixed distribution $\mathbb{P}$, returning an instance x and label $c(x)$ according to the unknown target concept
- Given access to EX, the learning algorithm outputs a Hypothesis $h \in \mathcal{H}$, which is a prediction rule on $\mathcal{X}$

[1] I'm going to use standard notation from computer science for this section. It will conflict with previous notation, but it quite firmly entrenched in the literature.

PAC-learning for classification

PAC-learning came about as a response to previous computer science attempts to find consistent learners,

(This is when we can perfectly classify a given set of instances $\leftrightarrow$ zero training error. These concept classes aren't truly interesting, as they are either trivial, impossible, and/or have super-polynomial growth in complexity)

The language of PAC-learning was developed to define whether a concept class $\mathcal{C}$ is learnable

Let $\text{err} = \mathbb{P}(h(x) \neq c(x))$

A concept class $\mathcal{C}$ is strongly learnable (with $\mathcal{H}$) provided there exists an algorithm A such that for all $\{c \in \mathcal{C}, \mathbb{P} \text{ on } \mathcal{X}, \epsilon > 0, \delta \leq 1\}$, there exists an $h \in \mathcal{H}$ where

$$\mathbb{P}^n(\text{err} \leq \epsilon) \geq \delta$$

(The number of instances from EX required grows $\leq$ polynomially in $1/\epsilon, 1/\delta$)

PAC-learning for classification

Example: Let

- The instances be $\mathcal{X} = \mathbb{R}$,
- The concept class $\mathcal{C}$ be the set of positive half lines

  (That is, an object that splits $\mathbb{R}$ into two pieces, labeling things to the left negative and the right positive)

The consistent learner would find any $h \in \mathcal{H} = \mathcal{C}$ in the transition region from negative to positive

- Is this a PAC learner?
- If so, how many examples do we need in order to ensure the learning algorithm is ϵ-good?

  (That is, $\text{err} \leq \epsilon$)
- Does the selection of the point in the transition region matter?

PAC-learning for classification

Example (continued): Fix an h and c. Let $h_*, c_* \in \mathbb{R}$ be the classification boundaries. Then, we only make a mistake if $x \in [c_* - h_*, c_* + h_*]$. Hence,

$$\text{err} = \mathbb{P}(h(x) \neq c(x)) = \mathbb{P}([c_* - h_*, c_* + h_*])$$

Note: From now on, think of h as being formed based on n data points $x_1, \ldots, x_n \overset{i.i.d}{\sim} \mathbb{P}$, and $\mathcal{D} = \{x_i\}_{i=1}^n$

PAC-learning for classification

Example (continued): Fix an $\epsilon > 0$.
Let g_+ be the minimal quantity such that $G_+ = [c_*, g_+]$ obeys $\mathbb{P}(G_+) \geq \epsilon$.

(Define G_- and g_- similarly for the other side of the interval)

Let's define two (bad) events

- $B_+ = (g_+, \infty)$
- $B_- = (-\infty, g_-)$

Intuition: If any $x \in \mathcal{D}$ falls in G_+, h_* will be in G_+ as

$$x \in G_+ \Rightarrow c(x) = +$$
$$x < c_* \Rightarrow c(x) = -$$

PAC-learning for classification

Example (continued):

By the previous argument, we can bound the probability of B_+

$$\begin{aligned}
\mathbb{P}^n(h_* \in B_+) &\leq \mathbb{P}^n(x_1 \notin G_+, \ldots, x_n \notin G_+) && \text{(product measure)} \\
&= \mathbb{P}(x_1 \notin G_+) \cdots \mathbb{P}(x_n \notin G_+) \\
&\leq (1-\epsilon)^n \\
&\leq e^{-\epsilon n} && (1+x \leq e^x)
\end{aligned}$$

PAC-learning for classification

Example (continued):

A concept class $\mathcal{C}$ is strongly learnable (with $\mathcal{H}$) provided there exists an algorithm A such that for all $\{c \in \mathcal{C}, \mathbb{P} \text{ on } \mathcal{X}, \epsilon > 0, \delta \leq 1\}$, there exists an $h \in \mathcal{H}$ where

$$\mathbb{P}^n(\text{err} \leq \epsilon) \geq \delta$$

$$\begin{aligned}
\mathbb{P}^n(\text{err} > \epsilon) &= \mathbb{P}^n(\mathbb{P}([c_* - h_*, c_* + h_*]) > \epsilon) \\
&\leq \mathbb{P}^n(\mathbb{P}([c_* - h_*]) > \epsilon \cup \mathbb{P}([c_* + h_*]) > \epsilon) \\
&\leq \mathbb{P}^n(\mathbb{P}([c_* - h_*]) > \epsilon) + \mathbb{P}^n(\mathbb{P}([c_* + h_*]) > \epsilon) \\
&\leq \mathbb{P}^n(h_* \in B_-) + \mathbb{P}^n(h_* \in B_+) \\
&\leq 2e^{-\epsilon n}
\end{aligned}$$

PAC-learning for classification

Example (conclusion):

(The number of instances from *EX* required grows $\leq$ polynomially in $1/\epsilon, 1/\delta$)

Let $\delta > 0$ be given.

$$\mathbb{P}^n(\mathrm{err} > \epsilon) \leq 2e^{-\epsilon n} \stackrel{set}{=} \delta$$

Result: $n > \epsilon^{-1} \log(2/\delta)$, this concept class is PAC-learnable

Alternatively, invert to find that with probability $1 - \delta$

$$\mathrm{err} \leq \frac{1}{n} \log(2/\delta)$$

PAC-learning for classification

A general result:

Suppose $|\mathcal{H}| < \infty$ (that is; we have a finite hypothesis space). If an algorithm A finds a hypothesis $h \in \mathcal{H}$ that is consistent with n examples, where

$$n > \frac{1}{\epsilon}(log(|\mathcal{H}|) + \log(1/\delta)).$$

Then

$$\mathbb{P}^n(\mathrm{err} > \epsilon) \leq \delta$$

Why does the bound depend on $|\mathcal{H}|$?

PAC-learning for classification

A general result:

Suppose $|\mathcal{H}| < \infty$ (that is; we have a finite hypothesis space). If an algorithm A finds a hypothesis $h \in \mathcal{H}$ that is consistent with n examples, where

$$n > \frac{1}{\epsilon}(log(|\mathcal{H}|) + \log(1/\delta)).$$

Then

$$\mathbb{P}^n(\text{err} > \epsilon) \leq \delta$$

Why does the bound depend on $|\mathcal{H}|$?

(The more rules, the more we can fit the training data (and then do worse predictions))

PAC-learning for classification

A general result:

Suppose $|\mathcal{H}| < \infty$ (that is; we have a finite hypothesis space). If an algorithm A finds a hypothesis $h \in \mathcal{H}$ that is consistent with n examples, where

$$n > \frac{1}{\epsilon}(log(|\mathcal{H}|) + \log(1/\delta)).$$

Then

$$\mathbb{P}^n(\text{err} > \epsilon) \leq \delta$$

Why does the bound depend on $|\mathcal{H}|$?

(The more rules, the more we can fit the training data (and then do worse predictions))

Why is the bound logarithmic in $|\mathcal{H}|$?

PAC-learning for classification

A general result:

Suppose $|\mathcal{H}| < \infty$ (that is; we have a finite hypothesis space). If an algorithm A finds a hypothesis $h \in \mathcal{H}$ that is consistent with n examples, where

$$n > \frac{1}{\epsilon}(log(|\mathcal{H}|) + \log(1/\delta)).$$

Then

$$\mathbb{P}^n(\text{err} > \epsilon) \leq \delta$$

Why does the bound depend on $|\mathcal{H}|$?

(The more rules, the more we can fit the training data (and then do worse predictions))

Why is the bound logarithmic in $|\mathcal{H}|$?

(Name each hypothesis in binary. How many bits do we need?

PAC-learning for classification

A general result:

Suppose $|\mathcal{H}| < \infty$ (that is; we have a finite hypothesis space). If an algorithm A finds a hypothesis $h \in \mathcal{H}$ that is consistent with n examples, where

$$n > \frac{1}{\epsilon}(log(|\mathcal{H}|) + \log(1/\delta)).$$

Then

$$\mathbb{P}^n(\text{err} > \epsilon) \leq \delta$$

Why does the bound depend on $|\mathcal{H}|$?

(The more rules, the more we can fit the training data (and then do worse predictions))

Why is the bound logarithmic in $|\mathcal{H}|$?

(Name each hypothesis in binary. How many bits do we need? $\log_2 |\mathcal{H}|$)

PAC-learning for classification

A general result (Proof):

$$
\begin{aligned}
\mathbb{P}^n(\text{err} > \epsilon) &= \mathbb{P}^n(\text{err} > \epsilon \cap h \text{ is consistent}) \\
&\leq \mathbb{P}^n(\text{err} > \epsilon \cap \exists h \in \mathcal{H} : h \text{ is consistent}) \\
&\leq \mathbb{P}^n(\exists h \text{ that is } \epsilon - \text{bad}, h \text{ is consistent}) \\
&\leq \sum_{h: h \text{ is } \epsilon-\text{bad}} \mathbb{P}^n(h \text{ is consistent}) \\
&= \sum_{h: h \text{ is } \epsilon-\text{bad}} \mathbb{P}^n(h(x_1) = c(x_1), \ldots, h(x_n) = c(x_n)) \\
&= \sum_{h: h \text{ is } \epsilon-\text{bad}} \prod_{i=1}^{n} \mathbb{P}(h(x_i) = c(x_i)) \\
&\leq \sum_{h: h \text{ is } \epsilon-\text{bad}} (1-\epsilon)^n
\end{aligned}
$$

PAC-learning for classification

A general result (Proof):

$$\begin{aligned}
\mathbb{P}^n(\text{err} > \epsilon) &\leq \sum_{h: h \text{ is } \epsilon-\text{bad}} (1-\epsilon)^n \\
&= |\{h : h \text{ is } \epsilon - \text{bad}\}|(1-\epsilon)^n \\
&\leq |\mathcal{H}|(1-\epsilon)^n \\
&\leq |\mathcal{H}|e^{-\epsilon n} \\
&\stackrel{set}{=} \delta
\end{aligned}$$

Invert to get result

The trick is that we're leveraging the finite nature of $\mathcal{H}$ to get a uniform bound

(This is why the set $\{h : h \text{ is } \epsilon - \text{bad}\}$ is nonrandom, it only depends on c, $\mathcal{H}$, $\mathbb{P}$, and ϵ. Tracking a particular consistent h is harder, as it is random)

PAC-learning for classification

What we need to refine this result for infinite $\mathcal{H}$ is to know more about its intrinsic complexity

A common notion for this is known as Vapnik-Chervonenkis (VC) dimension

This gives us a better idea of the complexity of the hypothesis space $\mathcal{H}$

VC-dimension

Shattering

Let $\mathcal{A}$ be a class of sets, such as

- $\mathcal{A} = \{(-\infty, t] : t \in \mathbb{R}\}$
- $\mathcal{A} = \{(a, b] : a \leq b\}$
- $\mathcal{A} =$ all rectangles in $\mathbb{R}^d$

Let $F = \{x_1, \ldots, x_n\}$ be a finite set, and $G \subseteq F$ (F is always finite)

We say that $\mathcal{A}$ picks out G (relative to F) if $\exists A \in \mathcal{A}$ s.t.

$$A \cap F = G$$

Example: Let $\mathcal{A} = \{(a, b] : a \leq b\}$, $F = \{1, 2, 7, 8, 9\}$ and $G = \{2, 7\}$. Then $\mathcal{A}$ picks out G (choose $A = (1.5, 7.5]$). However, $G = \{1, 9\}$ cannot be picked out by $\mathcal{A}$

Shattering

Let $S(\mathcal{A}, F)$ be the number of subsets of F that can be picked out by $\mathcal{A}$

Of course, $S(\mathcal{A}, F) \leq 2^n$

(The cardinality of the power set of F)

We say that F is shattered by $\mathcal{A}$ if $S(\mathcal{A}, F) = 2^n$

Also, let $\mathcal{F}_n$ be all finite sets with n elements

Then we have the shattering coefficient of $\mathcal{A}$

$$s_n(\mathcal{A}) = \sup_{F \in \mathcal{F}_n} S(\mathcal{A}, F)$$

Famous theorem: Let $\mathcal{A}$ be a class of sets. Then

$$\mathbb{P}(\sup_{A \in \mathcal{A}} |\hat{\mathbb{P}}(A) - \mathbb{P}(A)| > \epsilon) \leq 8 s_n(\mathcal{A}) e^{-n\epsilon^2/32}$$

(Vapnik, Chervonenkis (1971))

Shattering

This partly solves the problem. But, how big can $s_n(\mathcal{A})$ be?

Often times, $s_n(\mathcal{A}) = 2^n$ for all n up to some d, then $s_n(\mathcal{A}) < 2^n$ for all $n > d$

This d is the Vapnik-Chervonenkis (VC) dimension

Note: Often times, the subset formulation is converted to functions by assigning labels to the points.

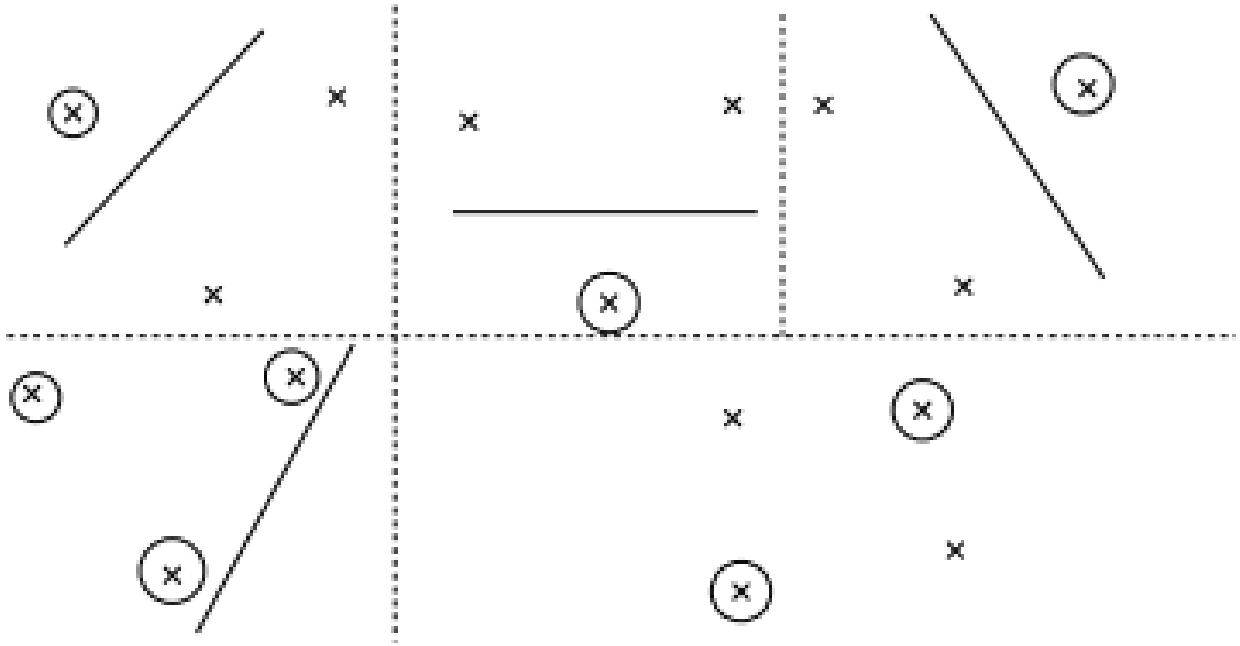
(This should be compared with SVMs, where we wish to separate points with hyperplanes)

VC-DIMENSION

Imagine that $\mathcal{A}$ is the set of hyperplanes in $\mathbb{R}^2$. Let $\mathcal{F}_n$ be all sets of n points.

We can shatter almost all $F \in \mathcal{F}_3$ (one is enough, though)

But, we cannot shatter any $F \in \mathcal{F}_4 \Rightarrow d = 3$

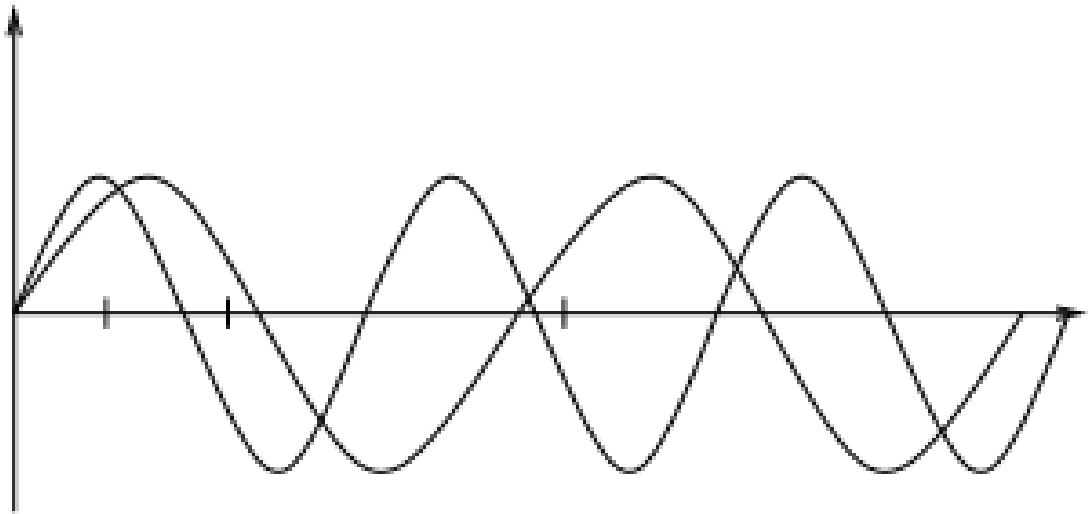


VC-dimension

It is tempting to think that the number of parameters determines the VC dimension

However, if we let $\mathcal{A} = \{\sin(tx) : t \in \mathbb{R}\}$, this is a one parameter family.

$\mathcal{A}$ can shatter a F for any $\mathcal{F}_n \Rightarrow d = \infty$



Bousquet et al. "Introduction to Statistical learning Theory"

VC-dimension and Sauer's theorem

Suppose that $\mathcal{A}$ has VC-dimension $d < \infty$. Then, for any $n \geq d$,

$$s_n(\mathcal{A}) \leq (n+1)^d$$

Punchline: For small n, the shattering coefficient increases exponentially. For large n, the shattering coefficient increases polynomially

If n is large enough and $d < \infty$ then

$$\begin{aligned}\mathbb{P}(\sup_{A \in \mathcal{A}} |\hat{\mathbb{P}}(A) - \mathbb{P}(A)| > \epsilon) &\leq 8 s_n(\mathcal{A}) e^{-n\epsilon^2/32} \\ &\leq 8(1+n)^d e^{-n\epsilon^2/32}\end{aligned}$$

(We'll leave this topic for now. We return weak learners/boosting and address VC dimension again soon.)

Boosting (again)

PAC-learning for classification

Suppose a learning algorithm cannot attain an error rate below a fixed amount (say 40%)

Can we still drive the error rate arbitrary close to zero?

Boosting considers this problem, augmenting classifiers that are only marginally better than random guessing

PAC-learning for classification

A concept class $\mathcal{C}$ is weakly learnable (with $\mathcal{H}$) provided there exists an algorithm A and $\gamma > 0$ such that for all $\{c \in \mathcal{C}, \mathbb{P} \text{ on } \mathcal{X}, \delta \leq 1\}$, there exists an $h \in \mathcal{H}$ produced by A on n examples where

$$\mathbb{P}^n(\text{err} \leq 1/2 - \gamma) \geq \delta$$

(Again, only polynomial growth of n is allowed)

Note: This means that there is an algorithm that, with high probability can do slightly better than random guessing

PAC-learning for classification

Example: Let $\mathcal{X} = \{0,1\}^n \cup \{Z\}$, $\mathcal{C}$ be all functions on $\mathcal{X}$, and $\mathbb{P}(\{Z\}) = 1/4$, uniform on all other elements. Given a set of examples, the algorithm will quickly learn $c(Z)$, as Z is very likely.

However, as we are only viewing polynomial number of examples from $|\mathcal{X}| \geq 2^n$, the algorithm will do not much better than random guessing. Then

$$\text{expected error} \approx \frac{1}{4}(0) + \frac{3}{4} \cdot \frac{1}{2} = \frac{3}{8}$$

So, there exists situations that are only weakly learnable. Is this the end of the story...?

Schapire (1990)

This paper answered the question: is there a gap between strong and weak learnability?

The answer is really no

A concept class $\mathcal{C}$ is weakly learnable if and only if it is strongly learnable[2]

The cool part is that the proof is constructive and produces the first boosting algorithm

[2]The wrinkle is that we are allowed to resample from altered versions of the training set and change the hypothesis class $\mathcal{H}$

Overall boosting philosophy

Given:

1. n examples from some fixed, unknown $\mathbb{P}$
2. a weak learning algorithm A producing $h \in \mathcal{H}$

Produce: a new hypothesis $H \in \mathcal{H}_{new}$ with error $\leq \epsilon$

Schapire (1990)

The first boosting algorithm for a weak learning algorithm A

1. A hypothesis h_1 is formed on n instances
2. A hypothesis h_2 is formed on n instances, half of which are misclassified by h_1

   (More specifically, a fair coin is flipped. If the result is heads then A draws samples $x \sim \mathbb{P}$ until $h_1(x) = c(x)$. If the result is tails then we wait until $h_1(x) \neq c(x)$)
3. A hypothesis h_3 is formed on n instances, for which h_1 and h_2 disagree
4. the boosted hypothesis h_b is the majority vote of h_1, h_2, h_3

Schapire's "strength of weak learnability" theorem shows that h_b has improved performance over h_1.

Schapire (1990)

Of course, if this rejection sampling is occurring over small probability regions of $\mathbb{P}$, then we may wait a long time (i.e.: not polynomially)

However, in the same paper they

1. bound the expected running time
2. use the δ parameter to bound with high probability the actual running time

(details omitted)

Freund (1995)

This paper augmented the results to show that we can combine many weak learners simultaneously

This improves the results of the simple boosting proposed by Schapire (1990)

The bottom line is that we can grow the number of comparisons sub-polynomially and still get polynomial complexity in the number of instances

Weakness of both papers

Each paper and associated theory required the weak learner to produce a classifier with a fixed error rate; that is γ

This led to a more realistic version, now known as AdaBoost

AdaBoost is more adaptive and realistic by dropping this assumption

AdaBoost

Boosting framework

Goal: Produce a sequence of (weak) classifiers $g_1, \ldots, g_M$ on repeatedly modified data

The final classifier is:

$$g(x) = \operatorname{sgn}\left(\sum_{m=1}^{M} \beta_m g_m(x)\right)$$

The β_m are the boosting weights to weight the classifiers

The modified data occurs by creating training weights $w_1, \ldots, w_n$ to weight the observations

AdaBoost outline

We give an overview of 'AdaBoost.M1.'

(Freund and Shapire (1997))

First, train the classifier as usual

(This is done by setting $w_i \equiv 1/n$)

At each step m, the misclassified observations have their weights increased

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

AdaBoost algorithm

1. Initialize $w_i \equiv 1/n$
2. For $m = 1, \ldots, M$
   - 2.1 Fit $g_m(x)$ on $\mathcal{D}$, weighted by w_i
   - 2.2 Compute

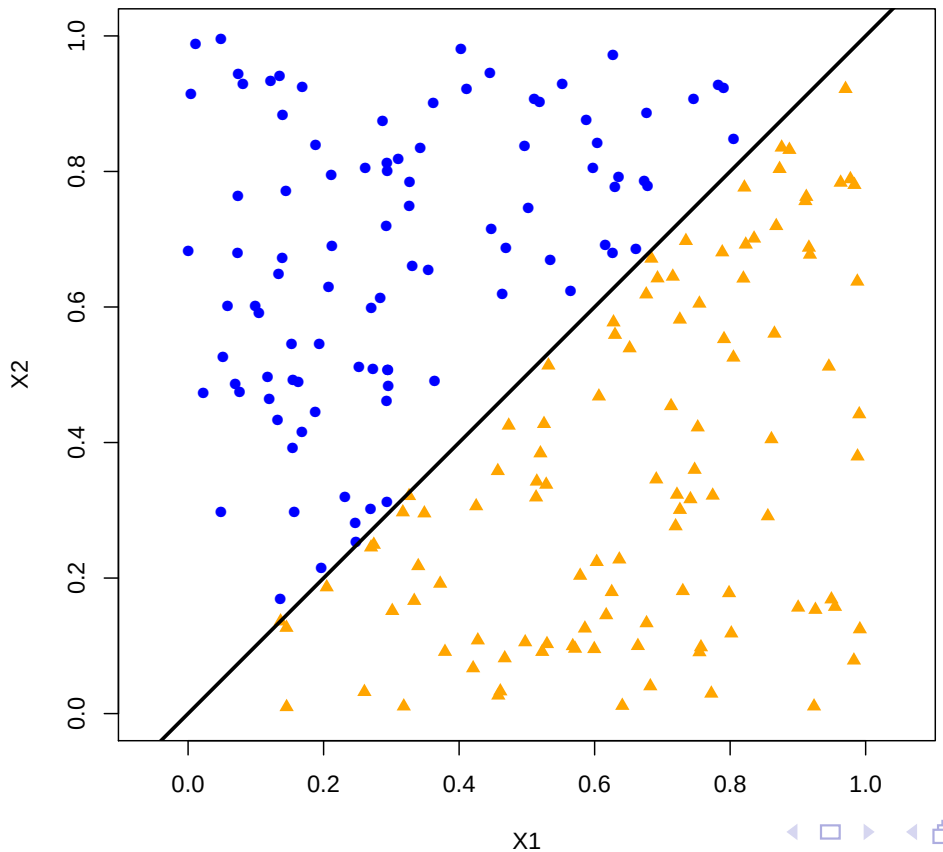
$$R_m = \frac{\sum_{i=1}^n w_i \mathbf{1}(Y_i \neq g_m(X_i))}{\sum_{i=1}^n w_i}$$

   - 2.3 Find $\beta_m = \log((1 - R_m)/R_m)$
   - 2.4 Set $w_i \leftarrow w_i \exp\{\beta_m \mathbf{1}(Y_i \neq g_m(X_i))\}$
3. Output: $g(x) = \operatorname{sgn}\left(\sum_{m=1}^M \beta_m g_m(x)\right)$

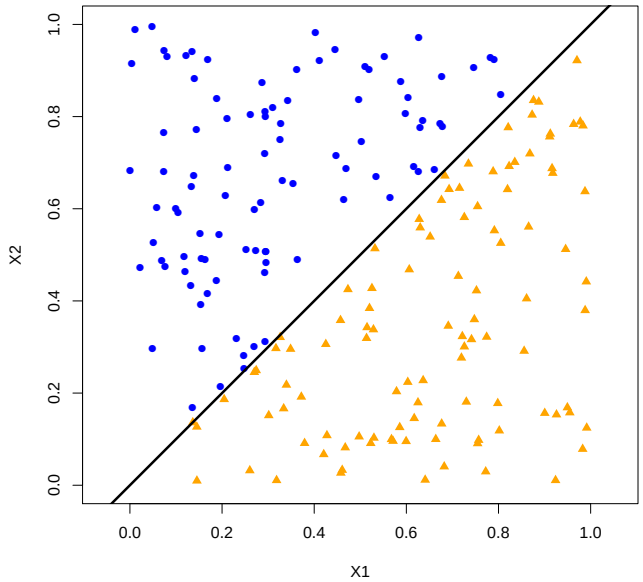
AdaBoost: Simulation

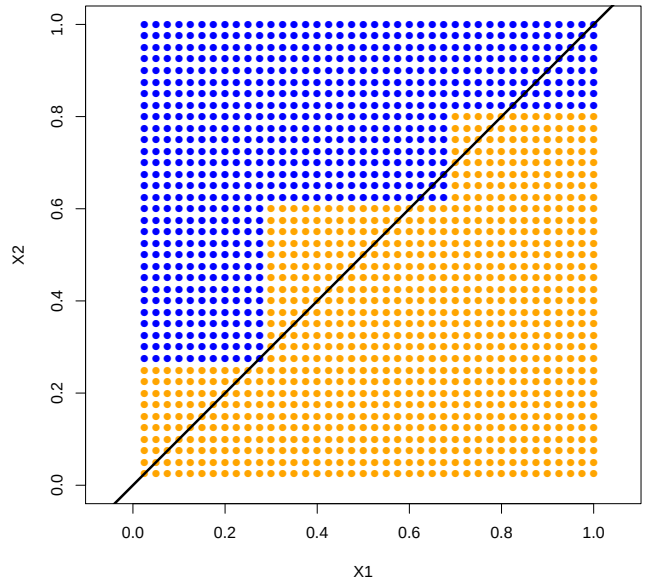
Let's use the classifier trees, but with 'depth 2-stumps'

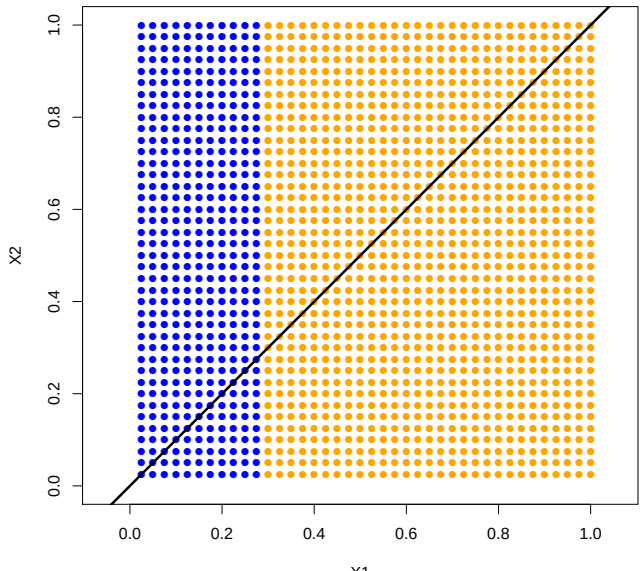
These are trees, but constrained to have no more than 4 terminal nodes

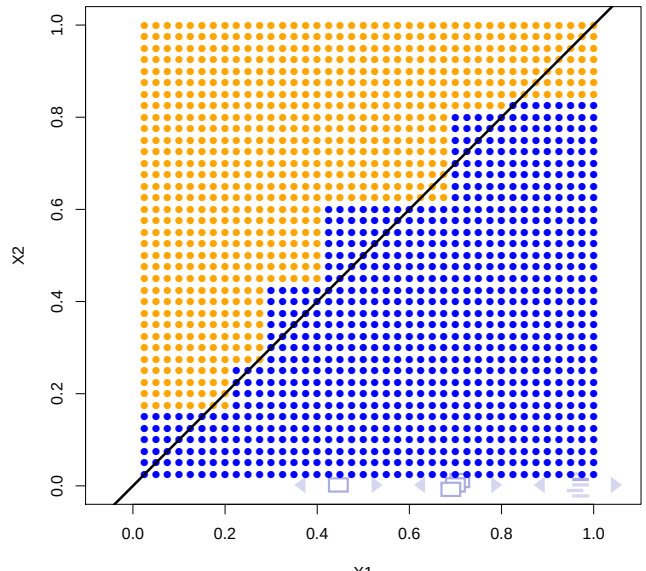


AdaBoost: Simulation

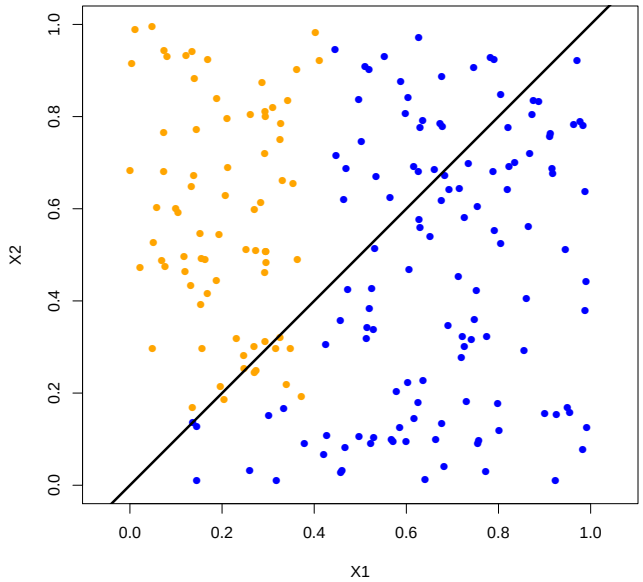


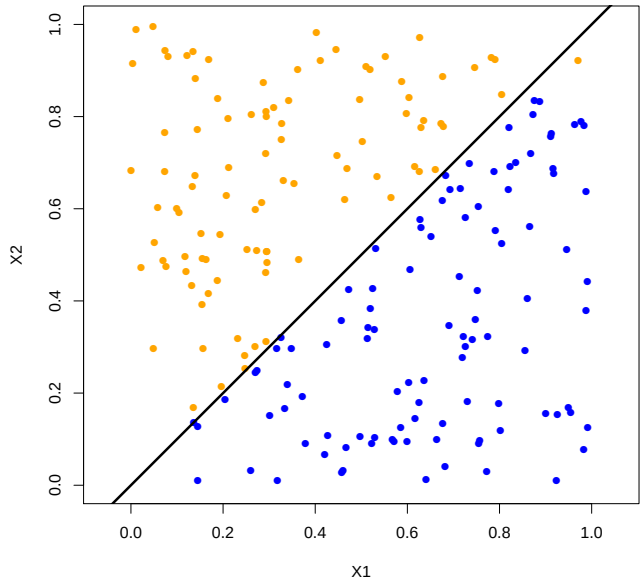


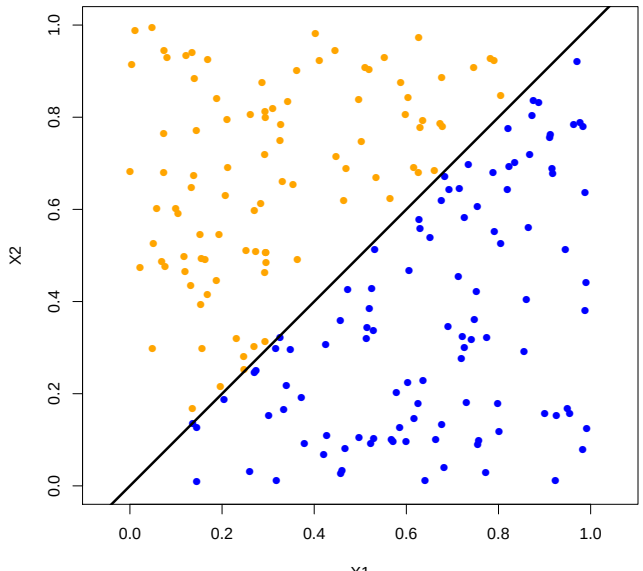


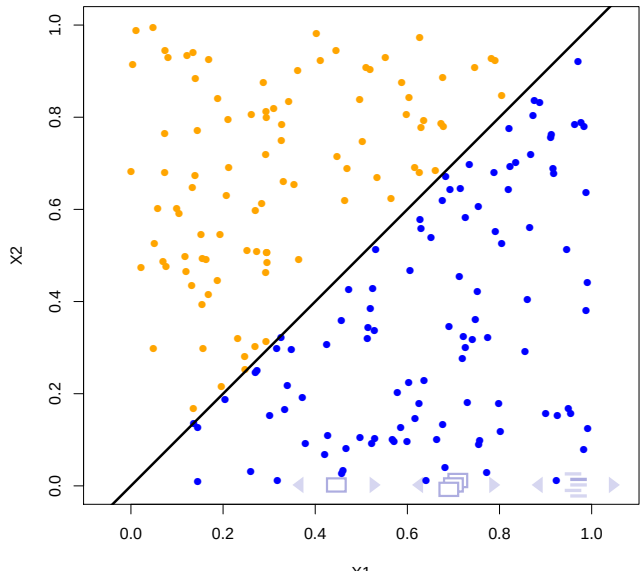


AdaBoost: Increasing M (train)

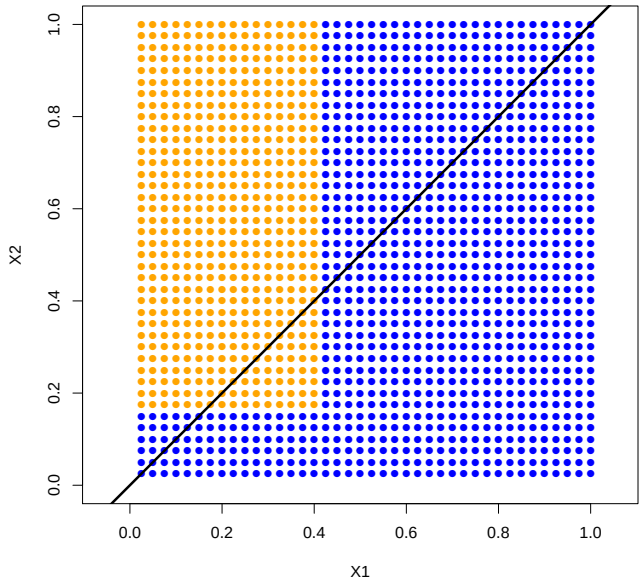


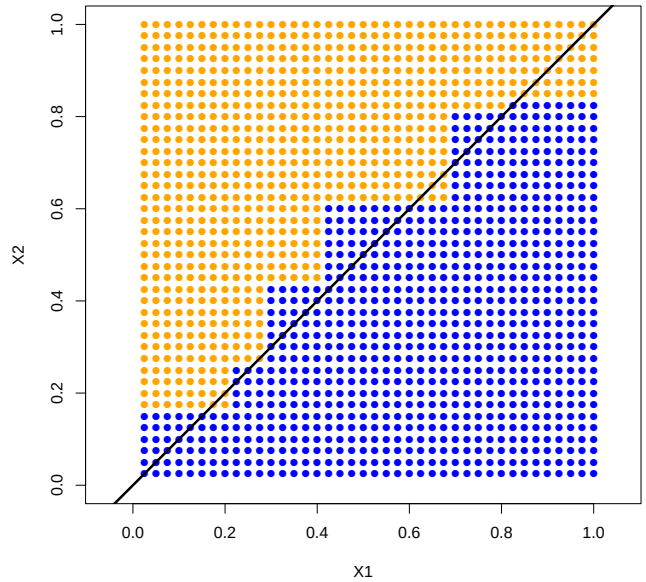


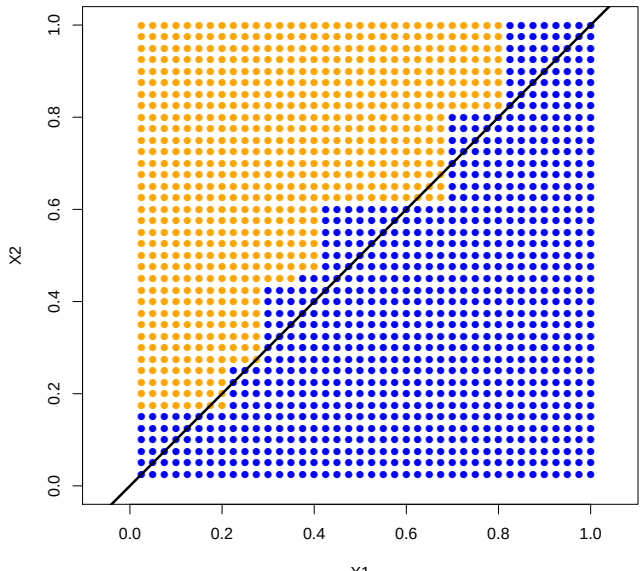


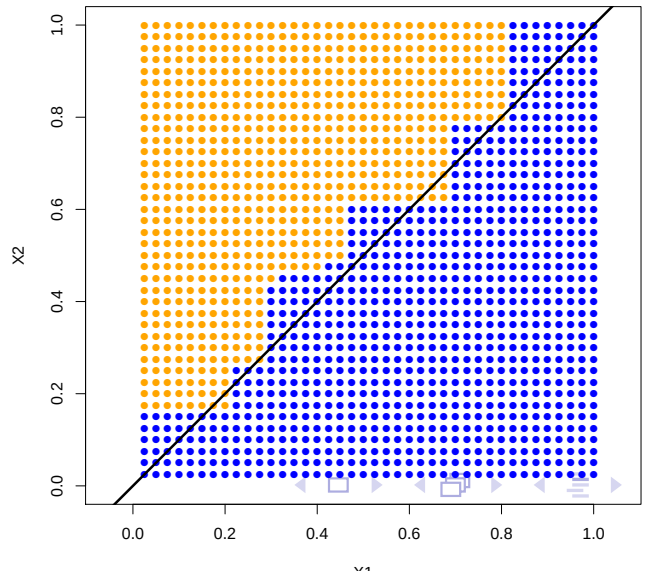


AdaBoost: Increasing M (test)



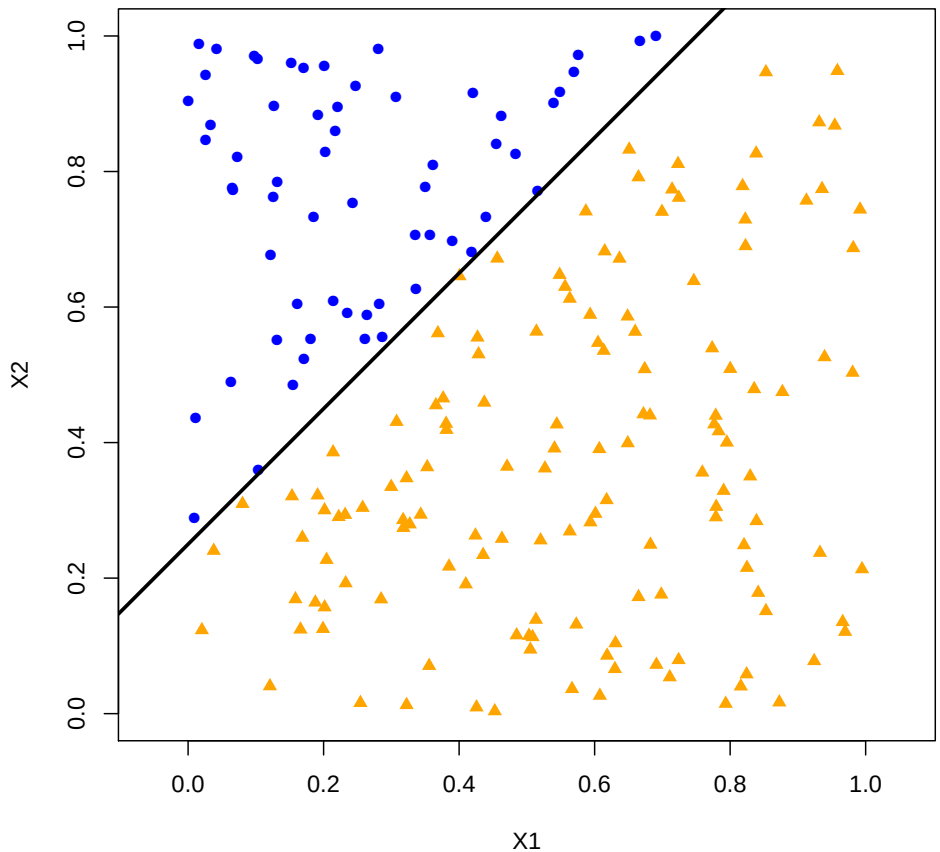




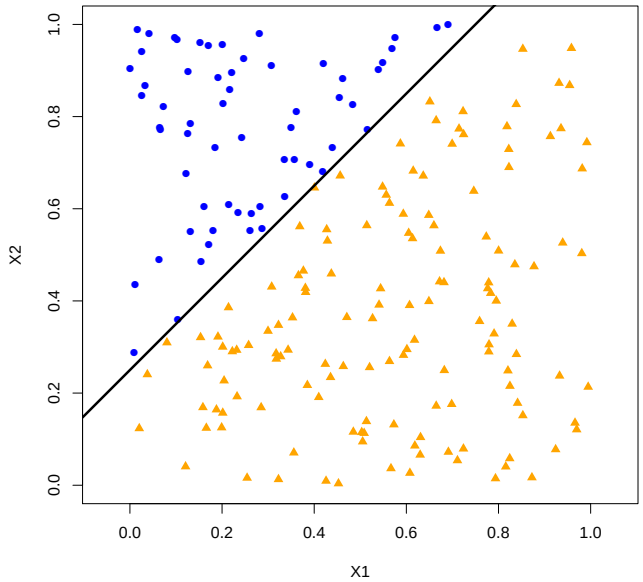


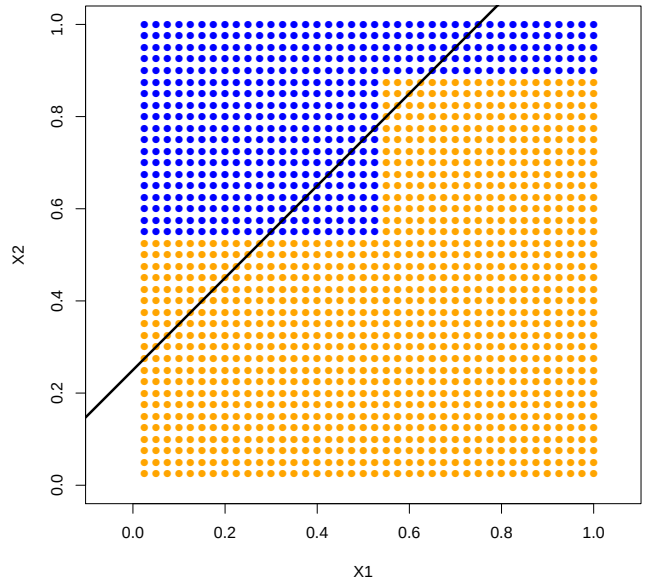
AdaBoost: Simulation

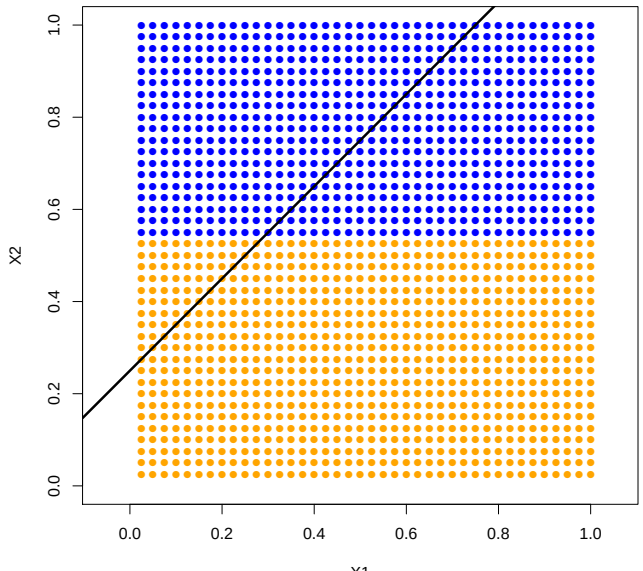
Let's change the simulation so that the class probabilities aren't the same

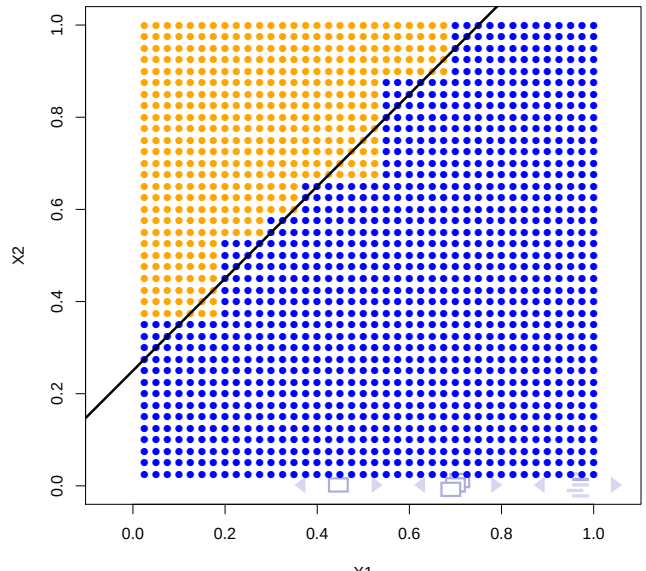


AdaBoost: Simulation

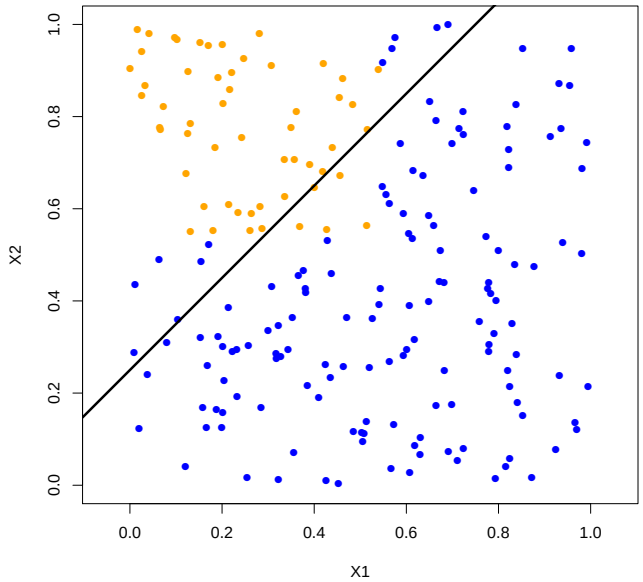


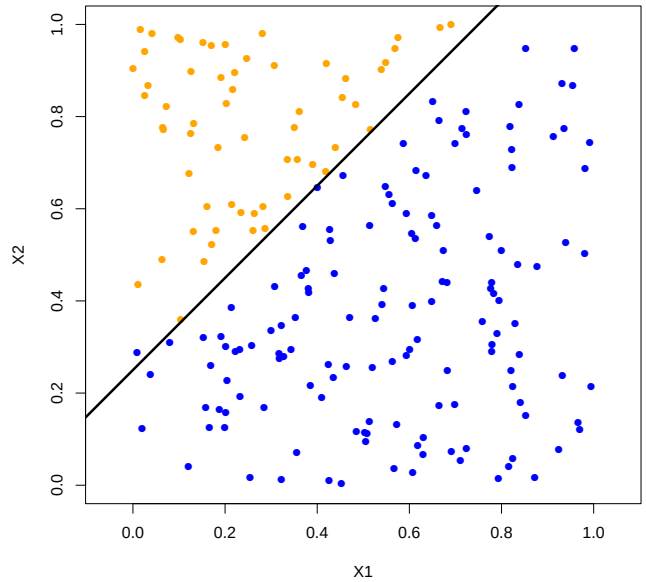


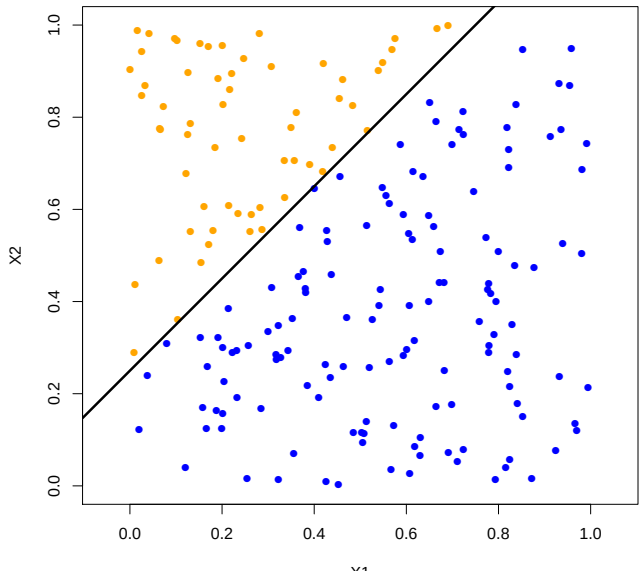


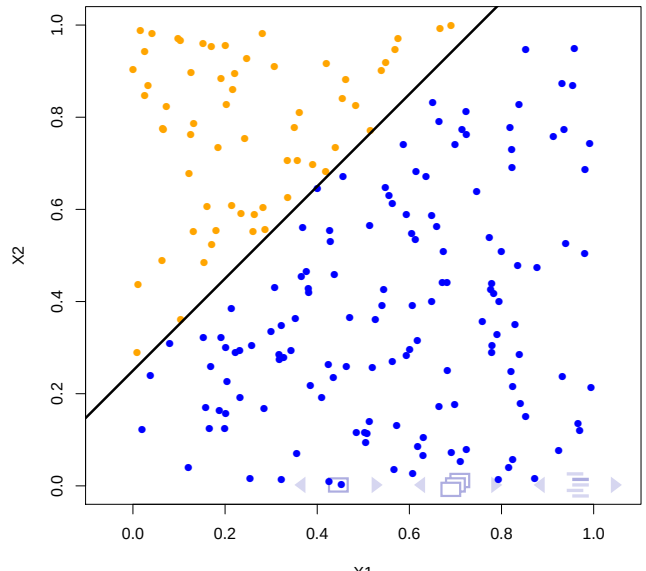


AdaBoost: Increasing M (train)

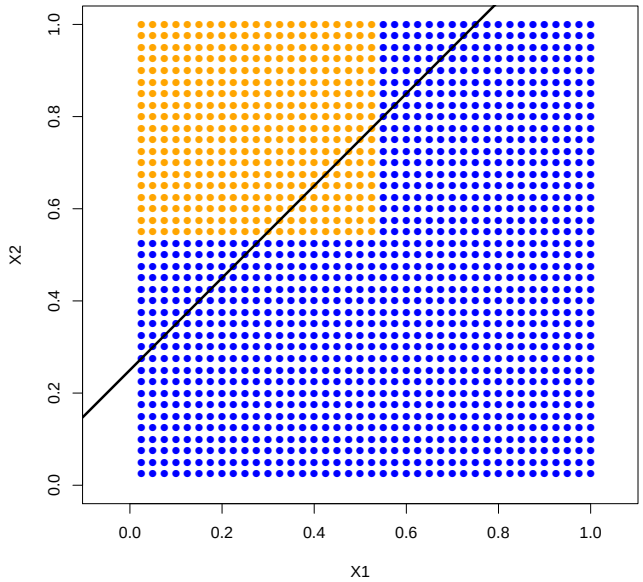


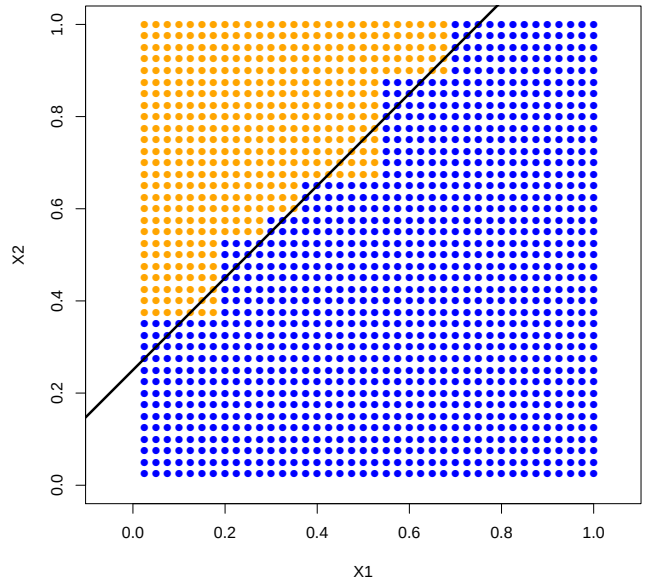


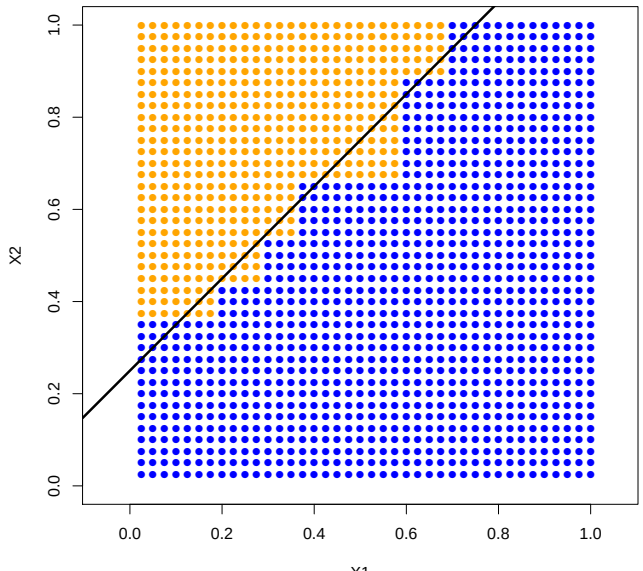


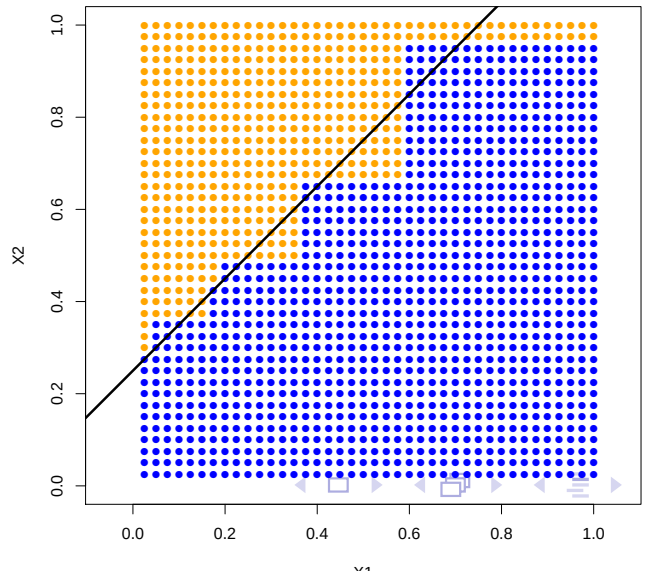


AdaBoost: Increasing M (test)









AdaBoost

This algorithm became known as ‘discrete AdaBoost’

This is due to the base classifier returning a discrete label

This was adapted to real-valued predictions in Real AdaBoost

(In particular, probability estimates)

Real AdaBoost

1. Initialize $w_i \equiv 1/n$
2. For $m = 1, \ldots, M$
   - 2.1 Fit the classifier on $\mathcal{D}$, weighted by w_i and produce $p_m(x) = \hat{P}_w(Y = 1|x)$
   - 2.2 Set $g_m(x) \leftarrow \frac{1}{2} \log(p_m/(1 - p_m(x)))$
   - 2.3 Set $w_i \leftarrow w_i \exp\{-Y_i g_m(X_i)\}$
3. Output: $g(x) = \text{sgn}\left(\sum_{m=1}^{M} g_m(x)\right)$

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

Real AdaBoost

1. Initialize $w_i \equiv 1/n$
2. For $m = 1, \ldots, M$
   - 2.1 Fit the classifier on $\mathcal{D}$, weighted by w_i and produce $p_m(x) = \hat{P}_w(Y = 1|x)$
   - 2.2 Set $g_m(x) \leftarrow \frac{1}{2} \log(p_m/(1 - p_m(x)))$
   - 2.3 Set $w_i \leftarrow w_i \exp\{-Y_i g_m(X_i)\}$
3. Output: $g(x) = \text{sgn}\left(\sum_{m=1}^{M} g_m(x)\right)$

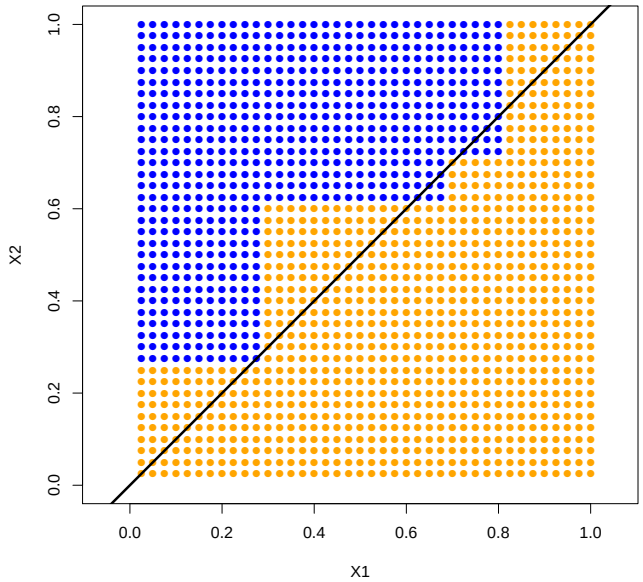
This is referred to as Real AdaBoost and it used the class probability estimates to construct the contribution of the m^{th} classifier, instead of the estimated label

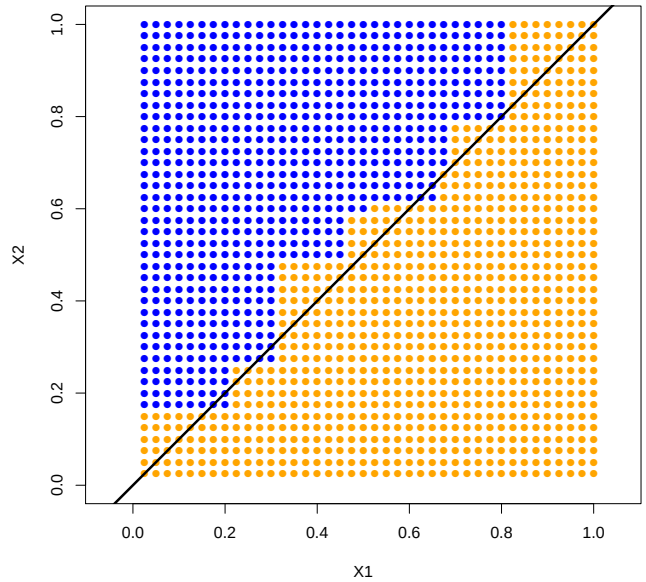
Real AdaBoost

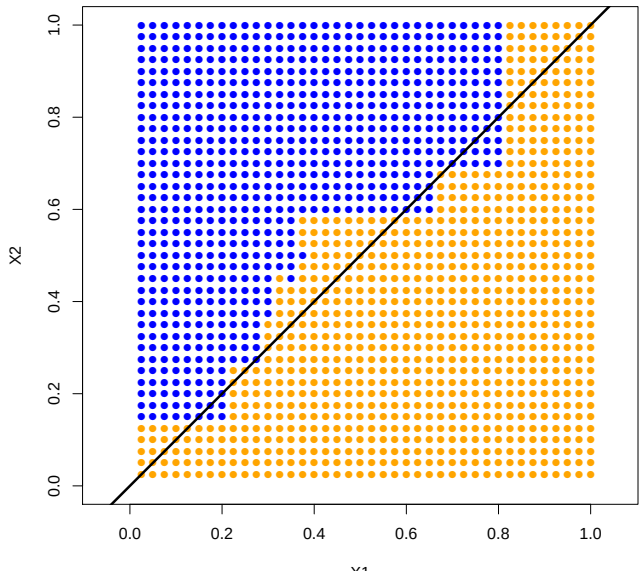
1. Initialize $w_i \equiv 1/n$
2. For $m = 1, \ldots, M$
   - 2.1 Fit the classifier on $\mathcal{D}$, weighted by w_i and produce $p_m(x) = \hat{P}_w(Y = 1|x)$
   - 2.2 Set $g_m(x) \leftarrow \frac{1}{2} \log((p_m + \epsilon)/(1 - p_m(x) + \epsilon))$
   - 2.3 Set $w_i \leftarrow w_i \exp\{-Y_i g_m(X_i)\}$
3. Output: $g(x) = \text{sgn}\left(\sum_{m=1}^{M} g_m(x)\right)$

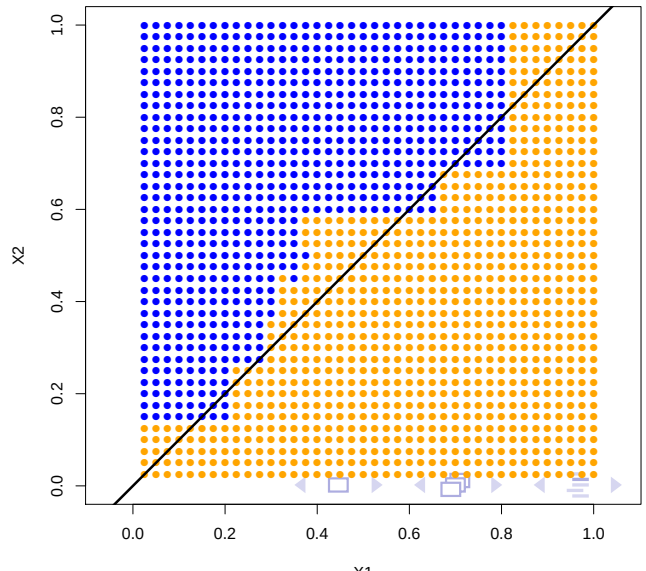
This is referred to as Real AdaBoost and it used the class probability estimates to construct the contribution of the m^{th} classifier, instead of the estimated label

AdaBoost: Increasing M (test)









AdaBoost intuition

Question: Why does this work?

One answer: Boosting fits an additive model

$$f(x) = \sum_{m=1}^{M} \beta_m \phi_m(x)$$

(It took about 5 years for this realization to appear in the literature)

Additive models

From linear to nonlinear models

Goal: Develop a prediction function $\hat{f} : \mathcal{X} \subset \mathbb{R}^p \to \mathbb{R}$ for predicting Y given an X

Commonly, $\hat{f}(X) = X^\top \beta$

(Constrained linear regression)

This greatly simplifies algorithms, while not sacrificing too much flexibility (think kernel methods)

However, sometimes directly modeling the nonlinearity is more natural

From linear to nonlinear models

Nonparametric methods form different types of local averages of the Y values of points near each other in $\mathcal{X}$

(Here, the meaning of 'near' gets specified by the method)

This works great if p is small (and the specification of nearness is good)

However, as p gets large

- nothing is nearby
- all points are on the boundary

  (Hence, predictions are generally extrapolations)

These features make up two components of the curse of dimensionality

(First usage: Bellman (1968) in the context of dynamic programming)

Curse of dimensionality

Fix the dimension p

(Assume p is even to ignore unimportant digressions)

Let S be a hypersphere with radius r

Let C be a hypercube with side length $2r$

Then, the volume of S and C are, respectively

$$V_S = \frac{r^p \pi^{p/2}}{(p/2)!} \text{ and } V_C = (2r)^p$$

(Interesting observation: this means for $r < 1/2$ the volume of the hypercube goes to 0, but the diagonal length is always $\propto \sqrt{p}$. Hence, the hypercube gets quite 'spiky' and is actually horribly jagged. Regardless of radius, the hypersphere's volume goes to zero quickly.)

Curse of dimensionality

Hence, the ratio of the volumes of a circumscribed hypersphere by a hypercube is

$$\frac{V_C}{V_S} = \frac{(2r)^p \cdot (p/2)!}{r^p \pi^{p/2}} = \frac{2^p \cdot (p/2)!}{\pi^{p/2}} = \left(\frac{4}{\pi}\right)^d d!$$

where $d = p/2$

Observation: This ratio of volumes is increasing really fast. This means that all of the volume of a hypersphere is near the corners. Also, this is independent of the radius.

Curse of dimensionality

This problem can be seen in the following table

The sample size required to ensure the MSE ≤ 0.1 (at 0) when the density is a multivariate normal is computed

(Silverman (1986). The method is kernel density estimation with optimal bandwidth)

Dimension	Sample Size
1	4
2	19
3	67
4	223
5	768
6	2790
7	10700
8	43700
9	187000
10	842000

Curse of dimensionality

Using minimax theory, we can further see the effect of dimension p

$$\inf_{\hat{f}} \sup_{f \in \Sigma_p(k,L)} \mathbb{E} \left|\left| \hat{f}(X) - f(X) \right|\right|_2^2 \asymp n^{-2k/(2k+p)}$$

(Here, $\Sigma_p(k, L)$ is the set of all functions whose k^{th} order partial derivatives are all L Lipchitz. See Gyorfi et al. (2002))

Let's invert this:

$$n \geq (1/\delta)^{(2k+p)/2k}$$

We need exponentially more observations to achieve a given minimax error level δ

Additive models

We can find a combination of linear models and nonlinear models that provides flexibility while shielding us somewhat from the dimension problem

Write

$$f(x) = f_1(x_1) + \cdots + f_p(x_p) = \sum_{j=1}^{p} f_j(x_j)$$

Estimation of such a function is not much more complicated than a fully linear model (as all inputs enter separately)

The algorithmic approach is known as backfitting

Additive models (for regression)

Additive models are usually phrased using the population level expectation

(These get replaced with empirical versions)

The update is a Gauss-Seidel-type update

(The Gauss-Seidel method is an iterative scheme for solving linear, square systems)

This is for $j = 1, \ldots, p, 1, \ldots, p, 1 \ldots$:

$$f_j(x_j) \leftarrow \mathbb{E}(Y - \sum_{k \neq j} f_k(x_k) | x_j)$$

Under fairly general conditions, this converges to the minimizer of $\mathbb{E}(Y - f(x))^2$

(See Buja et al. (1989))

Additive models (for regression)

Backfitting for additive models is roughly as follows:

Choose a univariate nonparametric smoother $\mathcal{S}$ and form all marginal fits $\hat{f}_j$

(Commonly a cubic smoothing spline with tuning parameter selected by GCV)

Iterate over j until convergence:

1. Define the residuals $R_i = Y_i - \sum_{k \neq j} \hat{f}_k(X_{ik})$
2. Smooth the residuals $\hat{f}_j = \mathcal{S}(R)$
3. Center $\hat{f}_j \leftarrow \hat{f}_j - \hat{\mathbb{P}}\hat{f}_j$

Report

$$\hat{f}(X) = \overline{Y} + \hat{f}_1(x_1) + \cdot + \hat{f}_p(x_p)$$

Additive models (for regression)

More generally, we can consider each function in the sum to be a function of all input variables

These functions (now indexed by m) are usually phrased as a base function $\phi_m(x) = \phi(x; \theta_m)$ and a multiplier β_m

Backfitting translates to iterating over:

$$\min_{\beta,\theta} \mathbb{E}\left[Y - \sum_{k \neq m} \beta_k \phi(x; \theta_k) - \beta\phi(x, \theta)\right]^2$$

(For most loss functions and basis function combinations minimizes jointly over all parameters is impossible)

Sometimes, even this is too burdensome, and is in need of a numerical approximation

Additive models (for regression)

In analogy to forward stepwise regression, we can do the minimization in a greedy fashion

(Remember: greedy means that at each step we don't revisit the fit from any previous step)

This is done by sequential minimization: For $m = 1, \ldots, M$

$$\beta_m, \theta_m = \underset{\beta,\theta}{\operatorname{argmin}} \mathbb{E}\left[Y - F_{m-1}(x) - \beta\phi(x, \theta)\right]^2$$

where $F_m(x) = \sum_{k=1}^{m-1} \beta_k \phi(x; \theta_k)$

For squared error loss, this reduces to finding the best single term basis expansion of the residuals

(This gives birth to the idea of least squares boosting. To bring this back to boosting, the F_m are known as the committee and the $\beta\phi(x; \theta)$ is the weak learner)

Detour: Signal Processing

This is the approach used in some signal processing-type applications. Mostly notably matching pursuit

(Mallat, Zhang (1993))

Matching pursuit forms an overcomplete dictionary of bases (e.g. wavelets and Fourier) that make up the $\phi(x; \theta)$,

(Here, θ indexes the scaling and location parameter modifying the mother wavelet and the frequency of the Fourier basis)

The aforementioned basis pursuit is the convex relaxation of this approach, which can provable exactly recover a sparse signal from an overcomplete dictionary as long as the basis vectors are sufficiently incoherent

(Chen et al. (1998))

(incoherence can be thought of correlation)

Additive models (for classification)

We learned from Bayes' theorem that all we need is the posterior class probabilities $\mathbb{P}(Y = j|X)$

We could transfer all the above regression machinery across by noting that this is the Bayes' rule under squared error

$$\mathbb{E}(\mathbf{1}(Y = j)|X) = \mathbb{P}(Y = j|X)$$

and hence use regression methods for estimating the posterior class probabilities

This works OK, but suffers from several problems, most notably the estimates aren't necessarily in $[0, 1]$

(A second, more subtle problem is called masking, which can occur when more than 2 classes are considered. See Hastie et al. (1994) for details.)

Additive models (for classification)

As squared error loss isn't quite right for classification, additive logistic regression is a popular approach

$$\log \frac{\mathbb{P}(Y=1|x)}{\mathbb{P}(Y=-1|x)} = \sum_{j=1}^{p} f_j(x_j) = F(x)$$

This gets inverted in the usual way to acquire a probability estimate

$$p(x) = \mathbb{P}(Y=1|x) = \frac{e^{F(x)}}{1+e^{F(x)}}$$

($F(x) = x^\top \beta$ gives us (linear) logistic regression)

These models are usually fit by numerically maximizing the binomial likelihood, and hence enjoy all the asymptotic optimality features of MLEs

Additive models (for classification)

In linear GLMs, the MLEs are found via Fisher scoring

At its core, we are finding updates $\mathbb{E}[\eta(X) + (Y - \mu)\frac{d\eta}{d\mu}|X]$

Given estimates $\hat{\eta} = \hat{\beta}^\top X$ and $\hat{\mu}$, we form working responses

$$Z = \hat{\eta} + (Y - \hat{\mu})\frac{d\eta}{d\mu}$$

and observational weights

$$W^{-1} = \left(\frac{d\eta}{d\mu}\right)^2 \mathbb{V}Y|_{\mu=\hat{\mu}}$$

(For logistic regression, $g(\mu) = \eta$, where g is the logistic function)

Iteratively regress Z on X with weights W, form $\hat{\mu}$, form Z,...

This is continued until the deviance doesn't change much

$$\text{dev}(Y, \hat{\mu}) = 2[\log(Y) - \log(\hat{\mu})]$$

Additive models (for classification)

Using this approach, we can form a generalized version of back fitting called local scoring

It looks like

1. Start with guesses $f_1(x_1), \ldots, f_p(x_p)$, $F(x) = \sum_{j=1}^{p} f_j(x_j)$ and $p(x)$
2. Form working responses

$$Z = F(x) + \frac{\mathbf{1}(Y = 1) - p(x)}{p(x)(1 - p(x))}$$

3. Apply back fitting to Z with observational weights $p(x)(1 - p(x))$ to produce $f_k(x_k)$

   (This is the data analogue to estimating $\mathbb{E}[\eta(X) + (Y - \mu)\frac{d\eta}{d\mu}|X]$, constrained only to an additive model)
4. Repeat until convergence

(See Hastie, Tibshirani (1986) for details. Note that this insight produces the LogitBoost algorithm.)

AdaBoost interpretation

Overall: Both discrete and real AdaBoost can be interpreted as stage wise estimation procedures for fitting additive logistic regression models

Rewriting forward stagewise additive modeling:

(Using a general likelihood ℓ and empirical expectation)

1. $\beta_m, \theta_m = \text{argmin}_{\beta,\theta} \sum_{i=1}^{n} \ell(Y_i, F_{m-1}(X_i) + \beta\phi(X_i, \theta))$
2. Set $F_m(x) = F_{m-1}(x) + \beta_m \phi(x; \theta_m)$

AdaBoost implicitly uses the loss function

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

and basis functions $\phi(x, \theta) = g_m(x)$

AdaBoost intuition

Suppose we minimize exponential loss in a forward stagewise manner

Doing the forward selection for this loss, we get

$$(\beta_m, g_m) = \underset{\beta, g}{\operatorname{argmin}} \sum_{i=1}^{n} \exp\{-Y_i(F_{m-1}(X_i) + \beta g(X_i))\}$$

AdaBoost intuition

Rewriting:

$$
\begin{aligned}
(\beta_m, g_m) &= \underset{\beta,g}{\operatorname{argmin}} \sum_{i=1}^{n} \exp\{-Y_i(F_{m-1}(X_i) + \beta g(X_i))\} \\
&= \underset{\beta,g}{\operatorname{argmin}} \sum_{i=1}^{n} \exp\{-Y_i F_{m-1}(X_i)\}\exp\{-Y_i \beta g(X_i))\} \\
&= \underset{\beta,g}{\operatorname{argmin}} \sum_{i=1}^{n} w_i \exp\{-Y_i \beta g(X_i)\}
\end{aligned}
$$

Where

- Define $w_i = \exp\{-Y_i F_{m-1}(X_i)\}$

  (This is independent of β, g)
- $\sum_{i=1}^{n} w_i \exp\{-Y_i \beta g_m(X_i))\}$ needs to be optimized

AdaBoost intuition

Note that

$$\sum_{i=1}^{n} w_i \exp\{-\beta Y_i g(X_i))\} = e^{-\beta} \sum_{i: Y_i = g(X_i)} w_i + e^{\beta} \sum_{i: Y_i \neq g(X_i)} w_i$$

$$= (e^{\beta} - e^{-\beta}) \sum_{i=1}^{n} w_i \mathbf{1}(Y_i \neq g(X_i)) +$$

$$+ e^{-\beta} \sum_{i=1}^{n} w_i$$

As long as $(e^{\beta} - e^{-\beta}) \geq 0$, we can find

$$g_m = \underset{g}{\text{argmin}} \sum_{i=1}^{n} w_i \mathbf{1}(Y_i \neq g(X_i))$$

(Note: If $(e^{\beta} - e^{-\beta}) < 0$, then $\beta < 0$. However, as $\beta_m = \log((1 - R_m)/R_m)$, this implies $R > 1/2$. Hence, we would flip the labels and get $R \leq 1/2$.)

Reminder: AdaBoost

1. Initialize $w_i \equiv 1/n$
2. For $m = 1, \ldots, M$
   - 2.1 Fit $g_m(x)$ on $\mathcal{D}$, weighted by w_i

     (This step is finding the next best version of the classifier, trained on weighted data and added to the previous classifiers)
   - 2.2 Compute

$$R_m = \frac{\sum_{i=1}^{n} w_i \mathbf{1}(Y_i \neq g_m(X_i))}{\sum_{i=1}^{n} w_i}$$

   - 2.3 Find $\beta_m = \log((1 - R_m)/R_m)$
   - 2.4 Set $w_i \leftarrow w_i \exp\{\beta_m \mathbf{1}(Y_i \neq g_m(X_i))\}$
3. Output: $g(x) = \operatorname{sgn}\left(\sum_{m=1}^{M} \beta_m g_m(x)\right)$

Reminder: AdaBoost

1. Initialize $w_i \equiv 1/n$
2. For $m = 1, \ldots, M$
   - 2.1 Fit $g_m(x)$ on $\mathcal{D}$, weighted by w_i

     (This step is finding the next best version of the classifier, trained on weighted data and added to the previous classifiers)
   - 2.2 Compute

$$R_m = \frac{\sum_{i=1}^n w_i \mathbf{1}(Y_i \neq g_m(X_i))}{\sum_{i=1}^n w_i}$$

   - 2.3 Find $\beta_m = \log((1 - R_m)/R_m)$
   - 2.4 Set $w_i \leftarrow w_i \exp\{\beta_m \mathbf{1}(Y_i \neq g_m(X_i))\}$
3. Output: $g(x) = \operatorname{sgn}\left(\sum_{m=1}^M \beta_m g_m(x)\right)$

AdaBoost intuition

Goal: Minimize

$$\sum_{i=1}^{n} w_i \exp\{-\beta Y_i g_m(X_i))\}$$

(Here, we have fixed $g = g_m$)

We showed this can be written

$$\sum_{i=1}^{n} w_i \exp\{-\beta Y_i g_m(X_i))\} = (e^{\beta} - e^{-\beta}) R_m W + e^{-\beta} W \quad (W = \sum w_i)$$

Take derivative with respect to β

$$(e^{\beta} + e^{-\beta}) R_m W - e^{-\beta} W \stackrel{set}{=} 0 \stackrel{set}{=} e^{\beta} R_m + e^{-\beta}(R_m - 1)$$

Solve for β to find $\beta_m = 1/2 \log[(1 - R_m)/R_m]$

Reminder: AdaBoost

1. Initialize $w_i \equiv 1/n$
2. For $m = 1, \ldots, M$
   - 2.1 Fit $g_m(x)$ on $\mathcal{D}$, weighted by w_i

     (This step is finding the next best version of the classifier, trained on weighted data and added to the previous classifiers)
   - 2.2 Compute

$$R_m = \frac{\sum_{i=1}^n w_i \mathbf{1}(Y_i \neq g_m(X_i))}{\sum_{i=1}^n w_i}$$

   - 2.3 Find $\beta_m = \log((1 - R_m)/R_m)$
   - 2.4 Set $w_i \leftarrow w_i \exp\{\beta_m \mathbf{1}(Y_i \neq g_m(X_i))\}$
3. Output: $g(x) = \operatorname{sgn}\left(\sum_{m=1}^M \beta_m g_m(x)\right)$

Reminder: AdaBoost

1. Initialize $w_i \equiv 1/n$
2. For $m = 1, \ldots, M$
   - 2.1 Fit $g_m(x)$ on $\mathcal{D}$, weighted by w_i

     (This step is finding the next best version of the classifier, trained on weighted data and added to the previous classifiers)
   - 2.2 Compute

$$R_m = \frac{\sum_{i=1}^{n} w_i \mathbf{1}(Y_i \neq g_m(X_i))}{\sum_{i=1}^{n} w_i}$$

   - 2.3 Find $\beta_m = \log((1 - R_m)/R_m)$
   - 2.4 Set $w_i \leftarrow w_i \exp\{\beta_m \mathbf{1}(Y_i \neq g_m(X_i))\}$
3. Output: $g(x) = \operatorname{sgn}\left(\sum_{m=1}^{M} \beta_m g_m(x)\right)$

AdaBoost intuition

The approximation is updated

$$F_m(x) = F_{m-1}(x) + \beta_m g_m(x)$$

This causes the weights

$$w_i^{(m+1)} = \exp\{-Y_i F_m(X_i)\} = w_i^{(m)} \exp\{-\beta_m Y_i g_m(X_i)\}$$

Using $Y_i g_m(X_i) = 2\mathbf{1}(Y_i \neq g_m(X_i)) - 1$, this becomes

$$w_i^{m+1} \propto w_i^m \exp\{\beta_m \mathbf{1}(Y_i \neq g_m(X_i))\}$$

where $\beta_m \leftarrow 2\beta_m$, giving the last step of the algorithm

AdaBoost: The controversy

Many attempts to explain boosting have come and gone

Each interpretation of boosting has provided insight, but ultimately cannot fully explain the empirical behavior

We will cover some of these ideas here, with the accompanying rebuttle

Exponential loss

Though the 'exponential loss' was motivated by very different principles originally, the main attraction is that it leads to a simple reweighting as above

However, it is interesting to ask about what it is estimating and how well?

An exponential criterion

Consider minimizing the criterion

$$J(F) = \mathbb{E}[e^{-YF(x)}|x]$$

(It is helpful to think of $\mathbb{E}$ as either the population level expectation, or the empirical expectation relative to a training sample, weighted by w. I'll try and write $\mathbb{E}_w$ when referring to the later)

For the population level expectation, the minimizer of $J(F)$ is the symmetric logistic transform of the posterior probabilities:

$$F(x) = \frac{1}{2} \log \frac{\mathbb{P}(Y = 1|x)}{\mathbb{P}(Y = -1|x)}$$

(Form $\mathbb{E}[e^{-YF(x)}] = \mathbb{P}(Y = 1|x)e^{-F(x)} + \mathbb{P}(Y = -1|x)e^{F(x)}$. Take Frechet derivative with respect to F and set equal to zero)

An exponential criterion

Therefore (after inverting)

$$F(x) = \frac{1}{2} \log \frac{\mathbb{P}(Y = 1|x)}{\mathbb{P}(Y = -1|x)}$$

$$\Downarrow$$

$$\mathbb{P}(Y = 1|x) = \frac{e^{F(x)}}{e^{-F(x)} + e^{F(x)}} \quad \text{and}$$

$$\mathbb{P}(Y = -1|x) = \frac{e^{-F(x)}}{e^{-F(x)} + e^{F(x)}}$$

Conclusion: Minimizing the exponential loss is equivalent (up to the 1/2 factor) to logistic regression. AdaBoost is estimating 1/2 the (conditional) log-odds of $Y = 1$

An exponential criterion

Another loss with the same population minimizer is the binomial (negative) log-likelihood

Define $\tilde{Y} = (Y+1)/2 \in \{0,1\}$. Then

$$\ell(Y, p(x)) = \tilde{Y} \log p(x) + (1-\tilde{Y}) \log(1-p(x))$$

Writing

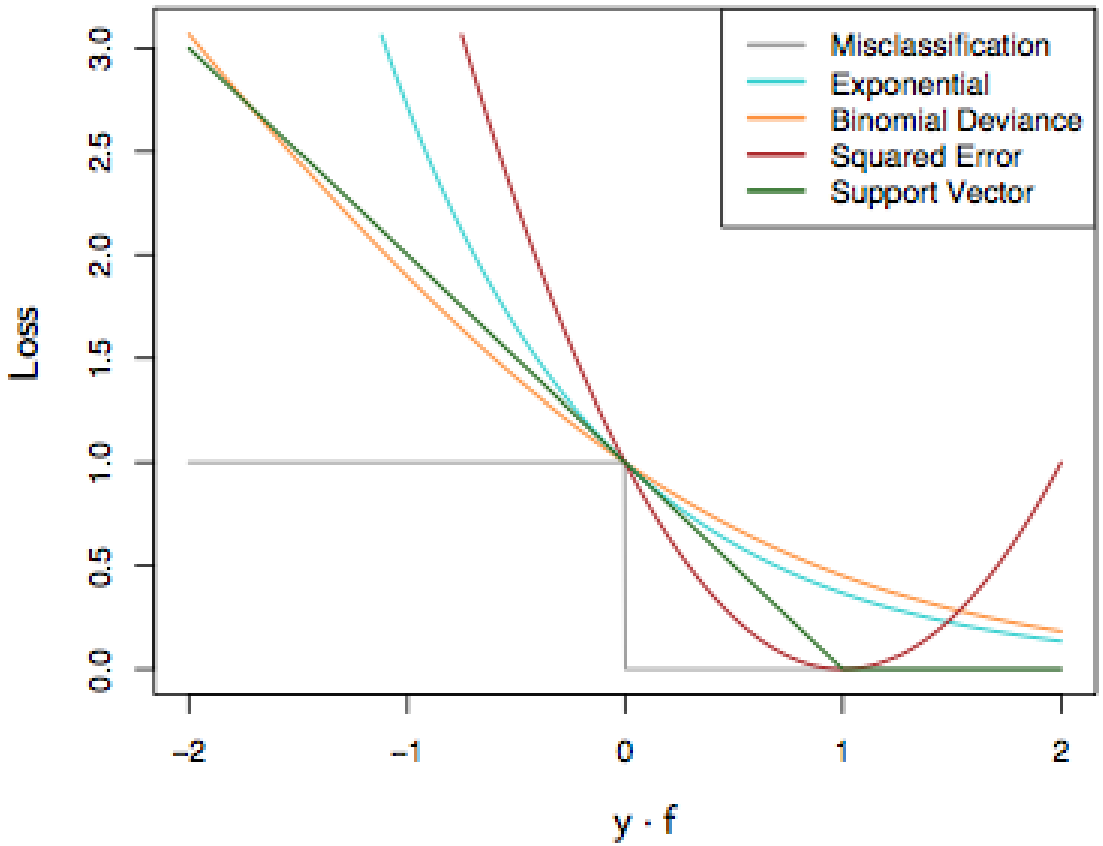
$$p(x) = \mathbb{P}(Y=1|x) = \frac{e^{F(x)}}{e^{-F(x)} + e^{F(x)}} = \frac{1}{1+e^{-2F(x)}}$$

and substituting, it shows that the deviance is

$$-\ell(Y, F(x)) = \log(1 + e^{-2YF(x)})$$

Hence, at a population level, the arg-minimizers are the same

Other loss functions



(Hastie et al (2009))

LogitBoost

1. Initialize $w_i \equiv 1/n$ and probability estimates $p(X_i) = 1/2$
2. For $m = 1, \ldots, M$
   - 2.1 Compute the working response and weights

$$Z_i = \frac{\tilde{Y}_i - p(X_i)}{p(X_i)(1 - p(X_i))}$$

$$w_i = p(X_i)(1 - p(X_i)$$

   - 2.2 Fit f_m by weighted least squares of Z_i on X with w_i
     (This is the same as the local scoring step)
   - 2.3 Update $F(x) \leftarrow F(x) + 1/2 f_m(x)$
   - 2.4 Update $p(x) \leftarrow e^{F(x)}/(e^{F(x)} + e^{-F(x)})$
3. Output: $g(x) = \text{sgn}\left(\sum_{m=1}^{M} f_m(x)\right)$

(Friedman, Hastie, Tibshirani (2000))

AdaBoost: The controversy

Claim: Boosting is another version of bagging

The early versions of Boosting involved (weighted) resampling

Therefore, it was initially speculated that a connection with bagging explained its performance

However, boosting continues to work well when

- The algorithm is trained on weighted data rather than on sampling with weights

  (This removes the randomization component that is essential to bagging)
- Weak learners are used that have high bias and low variance

  (This is the opposite of what is prescribed for bagging)

AdaBoost: The controversy

Claim: Boosting fits an adaptive additive model which explains its effectiveness

The previous results appeared in Friedman et al. (2000) and claimed to have 'solved' the mystery of boosting

A crucial property of boosting is that is essentially never over fits

However, the additive model view really should translate into intuition of 'over fitting is a major concern,' as it is with additive models

AdaBoost: The controversy

As adaBoost fits an additive model in the base classifier, it cannot have higher order interactions than the base classifier

For instance, a stump would provide a purely additive fit

(It only splits on one variable. In general, the complexity of a tree can be interpreted as the number of included interactions)

It stands to reason, then, if the Bayes' rule is additive in a similar fashion, stumps should perform well in Boosting

AdaBoost: The controversy

Assume that

$$\mathbb{P}(Y = 1|x) = q + (1 - 2q)\mathbf{1}\left(\sum_{j=1}^{J} x_j > J/2\right)$$

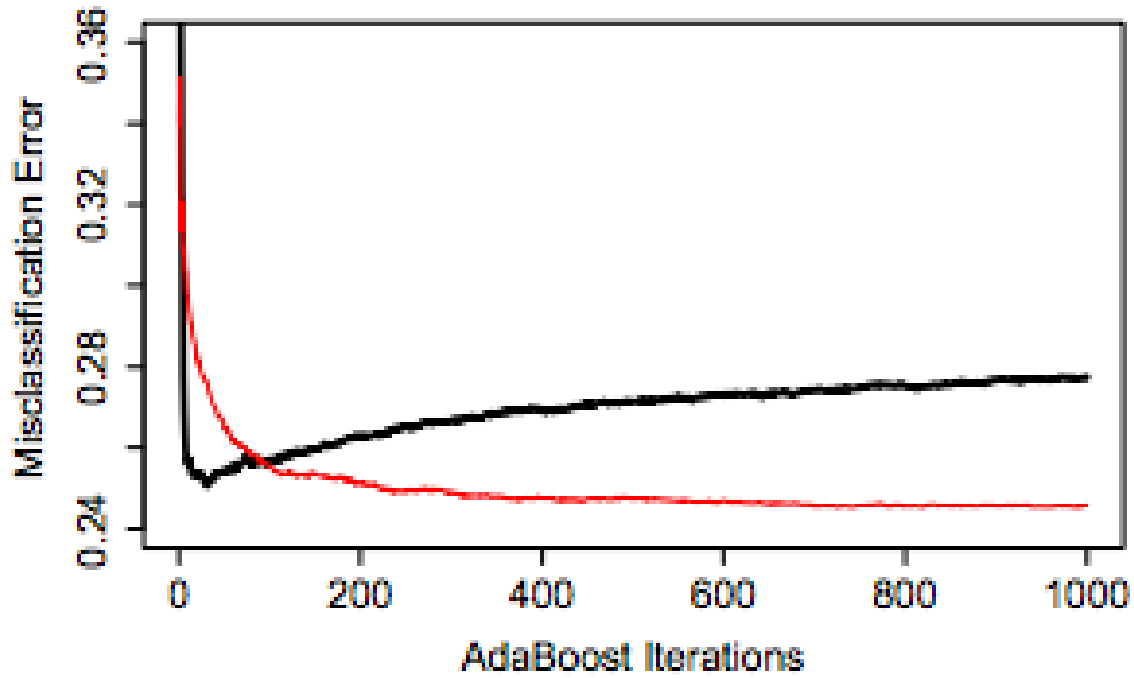


Figure: Black, bold line: Stumps. Red, thin line: 8-node trees

AdaBoost: The controversy continues

Ultimately, interpretations are just modes of human comprehension

The value of the insight is whether it provides fruitful thought about the idea

From this perspective, AdaBoost fits an additive model.

However, many of the other connections are still of debatable value

(For example, LogitBoost)

Next topic

Let $X_1, \ldots, X_n \sim \mathbb{P}$ such that $\mathbb{P}(\{0,1\}) = 1$ and $\theta = \mathbb{P}(X_1 = 1)$

We know that $\hat{\theta} = \overline{X}$ is MLE, UMVUE, blah

How good is it really?

We know $\hat{\theta} \sim (\theta, \theta(1-\theta)/n)$ and

$$\hat{\theta} \to \theta (a.s.)$$

$$\sqrt{n}(\hat{\theta} - \theta) \to \text{ Normal}$$

From this we can make confidence intervals

Next topic

It can be quite a bit more useful to have finite sample bounds for deviations from θ

A basic version of this is Hoëfding's inequality: let $S_n = \sum_{i=1}^{n} X_i$

$$\mathbb{P}(|S_n - n\theta| \geq t) \leq Ce^{-ct^2}$$

(C, c are universal constants)

Now, we know the distance to θ without any approximation..