Classification via Trees and Randomization

-Statistical Machine Learning-

Lecturer: Darren Homrighausen, PhD

An Introductory Example

Use macroeconomic data to predict recessions

Use handful of national-level variables – Federal Funds Rate, Term Spread, Industrial Production, Payroll Employment, S&P500

Also include state-level Payroll Employment

In this example, we code $Y = 1$ as a recession and $Y = 0$ as growth.

We will use data from 1960 through 1999 as training data

We will use data from 2000 through 2011 as testing data

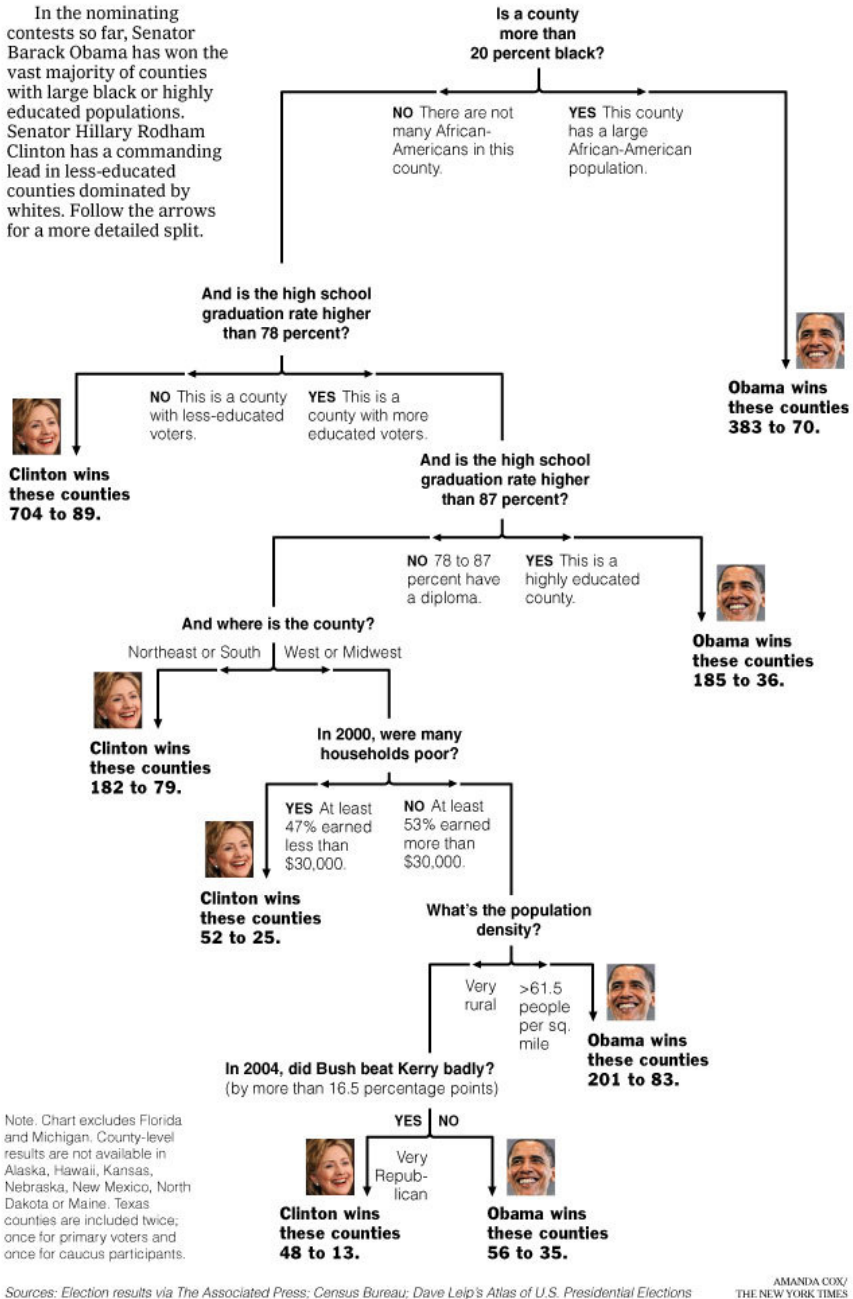
(See Owyang, Piger, Wall (2012))

Trees

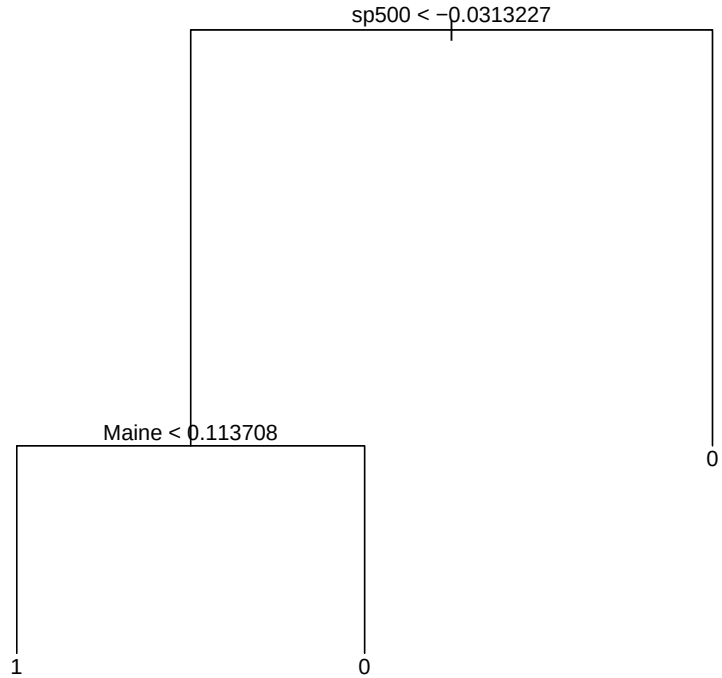
What is a (decision) tree?

- Trees involve stratifying or segmenting the predictor space into a number of simple regions.
- Trees are simple and useful for interpretation.
- Basic trees are not great at prediction.
- More modern methods that use trees are much better.

Example tree



DENDROGRAM VIEW



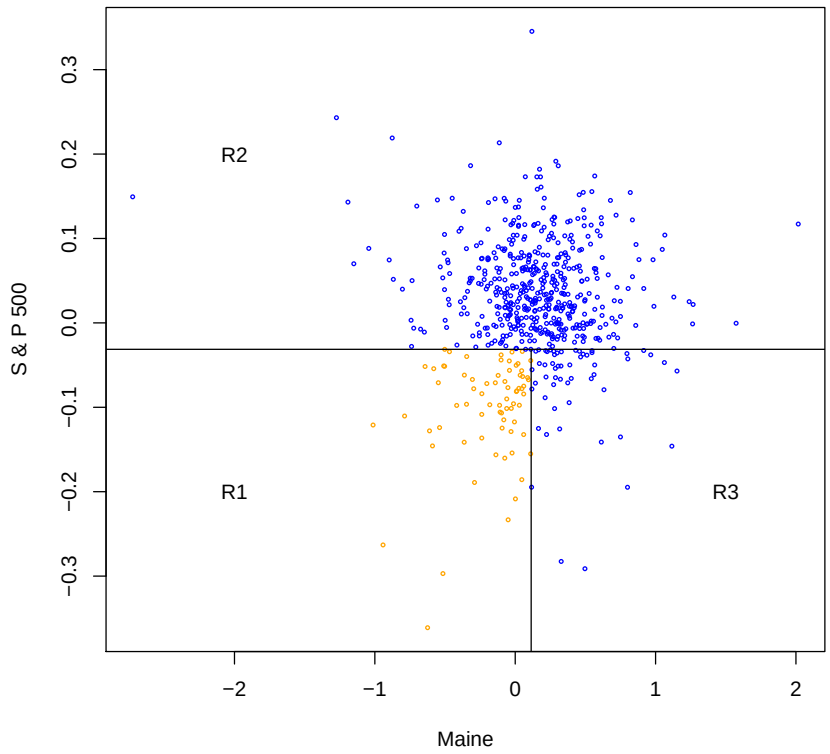
TERMINOLOGY

- We call each split or end point a node. Each terminal node is referred to as a leaf.

  This tree has 1 interior node and 3 terminal nodes.
- The interior nodes lead to branches.

  This graph has two main branches (the S&P 500 split).

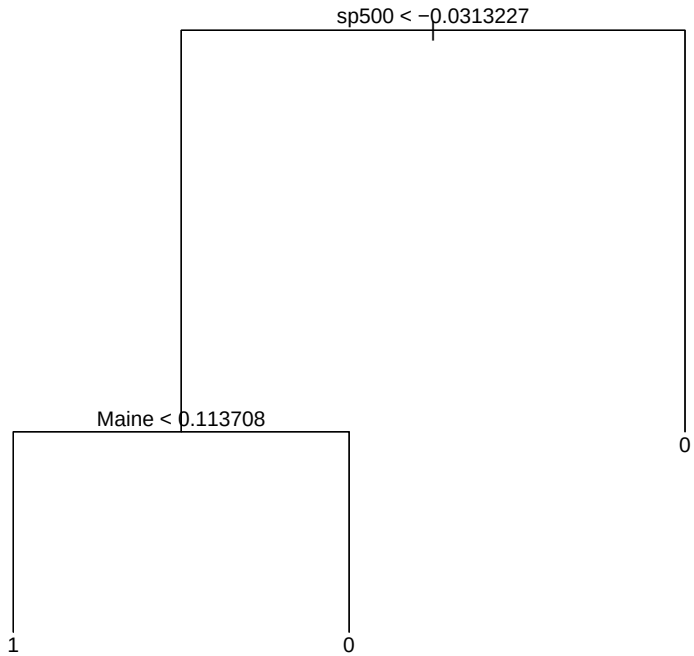
Partitioning view

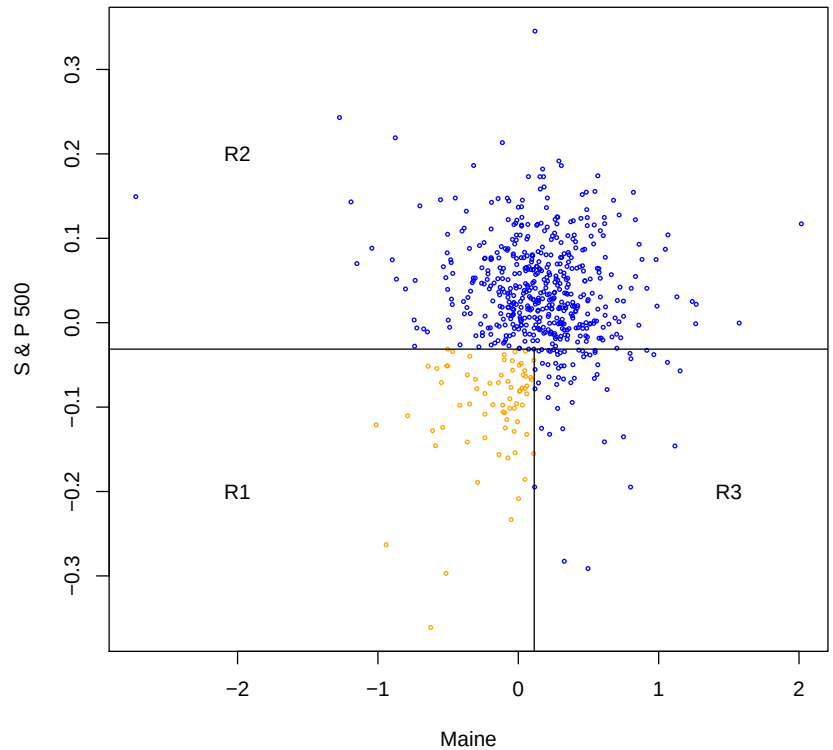


Notes

- We predict all observations in a region the same prediction
- The three regions R1, R2, and R3 are the terminal nodes

Tree





We can interpret this as (though this is with some reservation)

- S&P 500 is the most important variable.
- If S&P 500 is large enough, then we predict no recession.
- If S&P 500 is small enough, then we need to know the change in the employment level of Maine.

How do we build a tree?

1. Divide the predictor space into M non-overlapping regions $R_1, \ldots, R_M$

   (this is done via greedy, recursive, binary splitting)

2. Every observation that falls into a given region R_m is given the same prediction
   - Regression: The average of the responses for a region
   - Classification: Determined by majority (or plurality) vote in that region

Important:

- Trees can only make rectangular regions that are aligned with the coordinate axis.
- The fit is greedy, which means that after a split is made, all further decisions are conditional on that split.

Regression trees

Implicit model

For a given partition $R_1, \ldots, R_M$, the model for the response is

$$f(x) = \sum_{m=1}^{M} c_m \mathbf{1}_{R_m}(x)$$

For squared error loss, if $n_m = \sum_{i=1}^{n} \mathbf{1}_{R_m}(X_i)$, then

$$\hat{c}_m = n_m^{-1} \sum_{i : x_i \in R_m} Y_i$$

As for the regions, M encodes the tree complexity

This is challenging as considering all possible regions is computationally infeasible

(This would involve sifting through all $M \leq n$ and all configurations for R_m)

Model selection for trees

As a greedy approximation, do the following

1. **Grow a large tree:** $T_{\max}$, stopping when some minimal terminal node size requirement is met
2. **Cost-complexity pruning:** For all $\lambda \geq 0$

$$C_\lambda(T) = \sum_{m=1}^{M} \sum_{i:x_i \in R_m} (Y_i - \hat{c}_m)^2 + \lambda M$$

(Note that often it is written that $|T| = M$)

3. **Weakest link pruning:** For each λ, there is a unique smallest T_λ that minimizes $C_\lambda(T)$. Eliminating nodes that produce the smallest increase in training error produces a sequence of solutions that must contain T_λ

(many details ommitted)

Classification trees

Classification trees

The only modification for classification is choice of loss function

For region m and class g, we get training proportions

$$\hat{p}_{mg}(x) = \mathbf{1}_{R_m}(x) n_m^{-1} \sum_{i:X_i \in R_m} \mathbf{1}(Y_i = g)$$

Our classification is

$$\hat{g}(x) = \arg\max_g \hat{p}_{mg}(x)$$

How do we measure quality of fit?

Different measures of node impurity (loss function in tree terminology)

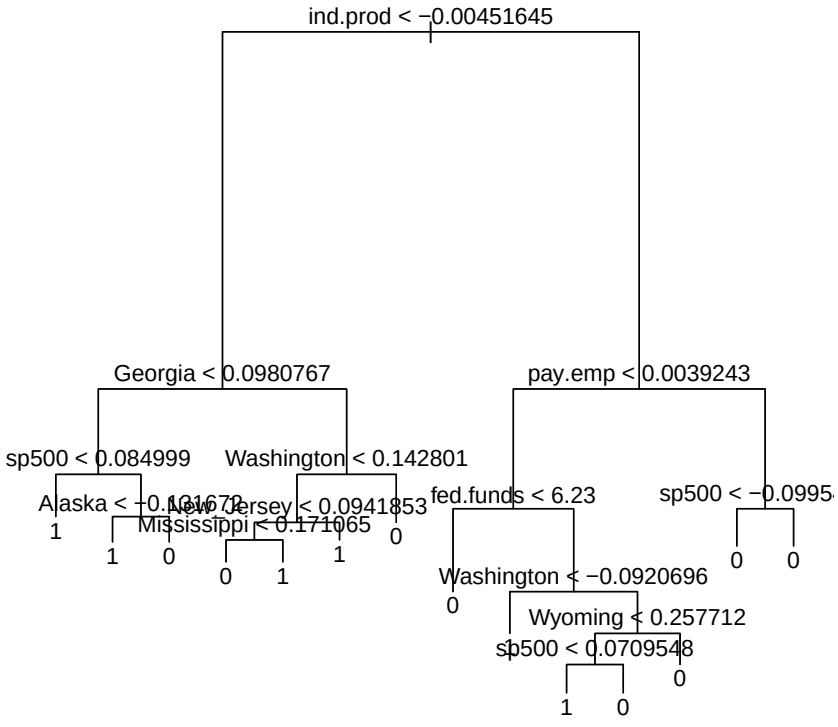
classification error rate:	$E = 1 - \max_k(\hat{p}_{mk})$
Gini index:	$G = \sum_k \hat{p}_{mk}(1 - \hat{p}_{mk})$
cross-entropy:	$D = -\sum_k \hat{p}_{mk} \log(\hat{p}_{mk})$

Both G and D can be thought of as measuring the purity of the classifier (small if all $\hat{p}_{mk}$ are near zero or 1). These are preferred over the classification error rate.

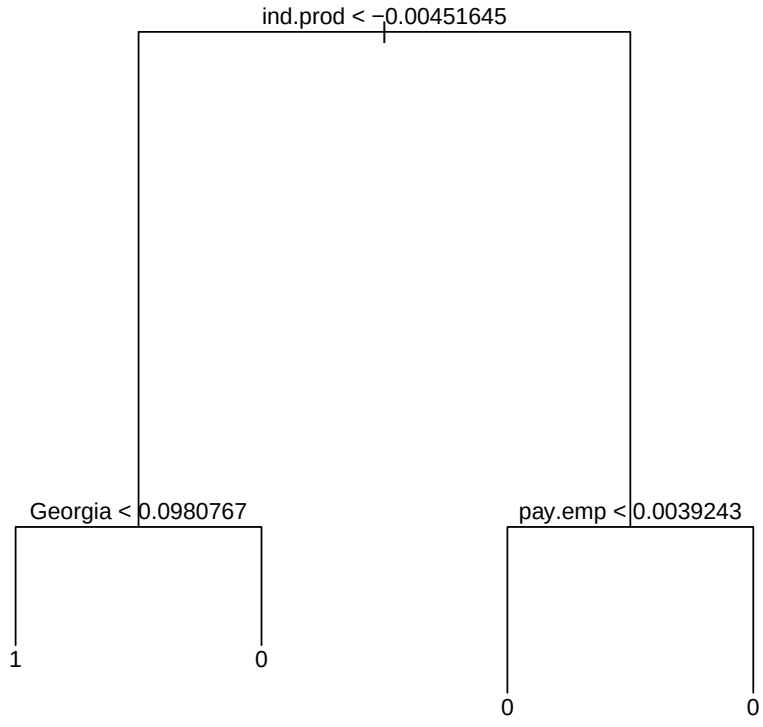
(Also, E isn't differentiable and hence not as amenable to numerical optimization)

We build a classifier by growing a tree that minimizes G or D.

Results of trees on recession data



Unpruned tree



Pruned Tree

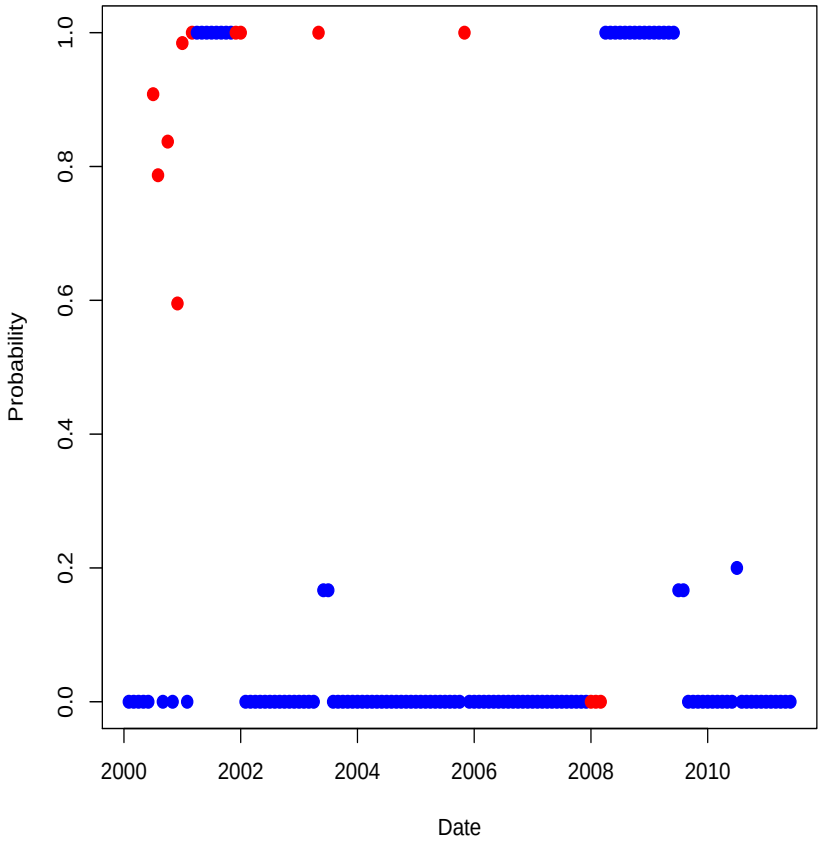
(Tuning parameter chosen by cross-validation)

Results of trees on recession data

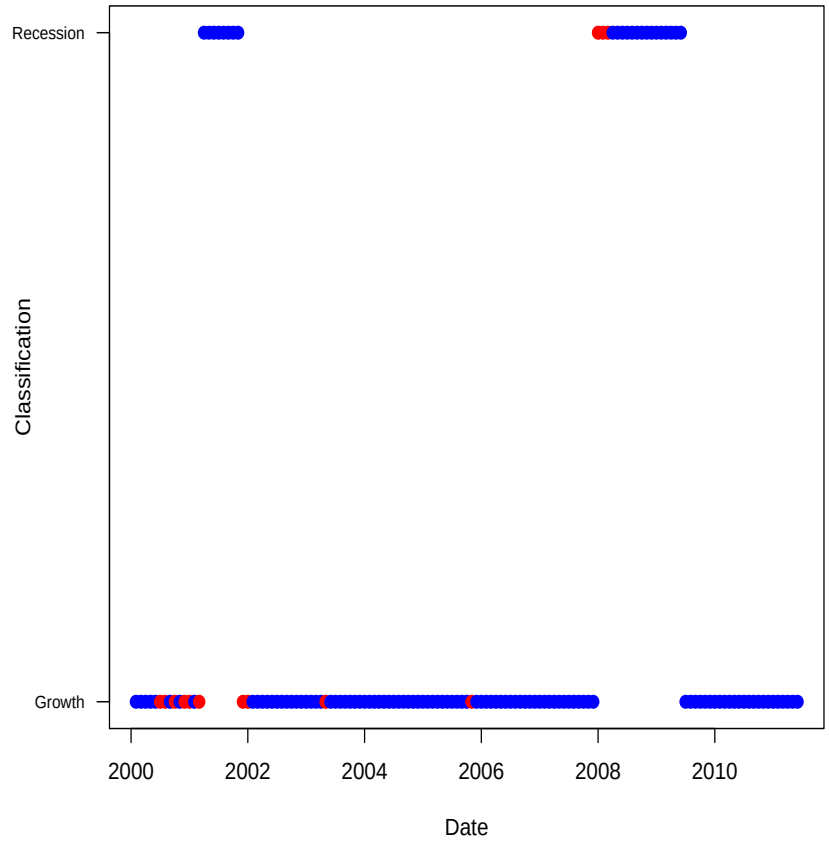
The trees on the previous slide were grown on the training data (from 1960 to 2000)

Now, we use them to predict on the test data (from 2000 to 2011)

Results of trees on recession data



Posterior probability of prediction



Predictions

Advantages and disadvantages of trees

- \+ Trees are very easy to explain (much easier than even linear regression).
- \+ Some people believe that decision trees mirror human decision.
- \+ Trees can easily be displayed graphically no matter the dimension of the data.
- \+ Trees can easily handle qualitative predictors without the need to create dummy variables.
- − Trees aren't very good at prediction.

To fix this last one, we can try to grow many trees and average their performance.

Bagging

Bagging

Many methods (trees included) tend to be designed to have lower bias but high variance

This means that if we split the training data into two parts at random and fit a decision tree to each part, the results could be quite different

A low variance estimator would yield similar results if applied repeatedly to distinct data sets

(consider $\hat{f}(\mathbf{x}) = 0$ for all $\mathbf{x}$)

Bagging, also known as Bootstrap AGgregation, is a general purpose procedure for reducing variance.

We'll use it specifically in the context of trees, but it can be applied more broadly.

Bagging: The main idea

Suppose we have n uncorrelated observations $Z_1, \ldots, Z_n$, each with variance σ^2.

What is the variance of

$$\overline{Z} = \frac{1}{n} \sum_{i=1}^{n} Z_i ?$$

Bagging: The main idea

Suppose we have n uncorrelated observations $Z_1, \ldots, Z_n$, each with variance σ^2.

What is the variance of

$$\overline{Z} = \frac{1}{n} \sum_{i=1}^{n} Z_i ?$$

Answer: σ^2/n.

More generally, if we have B separate (uncorrelated) training sets, $1, \ldots, B$, we can form B separate model fits, $\hat{f}^1(\mathbf{x}), \ldots, \hat{f}^B(\mathbf{x})$, and then average them:

$$\hat{f}_B(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^b(\mathbf{x})$$

Bagging: The bootstrap part

Of course, this isn't practical as having access to many training sets is unlikely.

We therefore turn to the bootstrap to simulate having many training sets.

The bootstrap is a widely applicable statistical tool that can be used to quantify uncertainty without Gaussian approximations.

Let's look at an example.

Bootstrap detour

Bootstrap detour

Suppose we are looking to invest in two financial instruments, X and Y. The return on these investments is random, but we still want to allocate our money in a risk minimizing way.

That is, for some $\alpha \in (0, 1)$, we want to minimize

$$\mathrm{Var}(\alpha X + (1 - \alpha) Y)$$

The minimizing α is:

$$\alpha_* = \frac{\sigma_Y^2 - \sigma_{XY}{}^2}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}{}^2}$$

which we can estimate via

$$\hat{\alpha} = \frac{\hat{\sigma}_Y^2 - \hat{\sigma}_{XY}^2}{\hat{\sigma}_X^2 + \hat{\sigma}_Y^2 - 2\hat{\sigma}_{XY}^2}$$

Bootstrap detour

Suppose we are looking to invest in two financial instruments, X and Y. The return on these investments is random, but we still want to allocate our money in a risk minimizing way.

That is, for some $\alpha \in (0, 1)$, we want to minimize

$$\mathrm{Var}(\alpha X + (1 - \alpha) Y)$$

The minimizing α is:

$$\alpha_* = \frac{\sigma_Y^2 - \sigma_{XY}{}^2}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}{}^2}$$

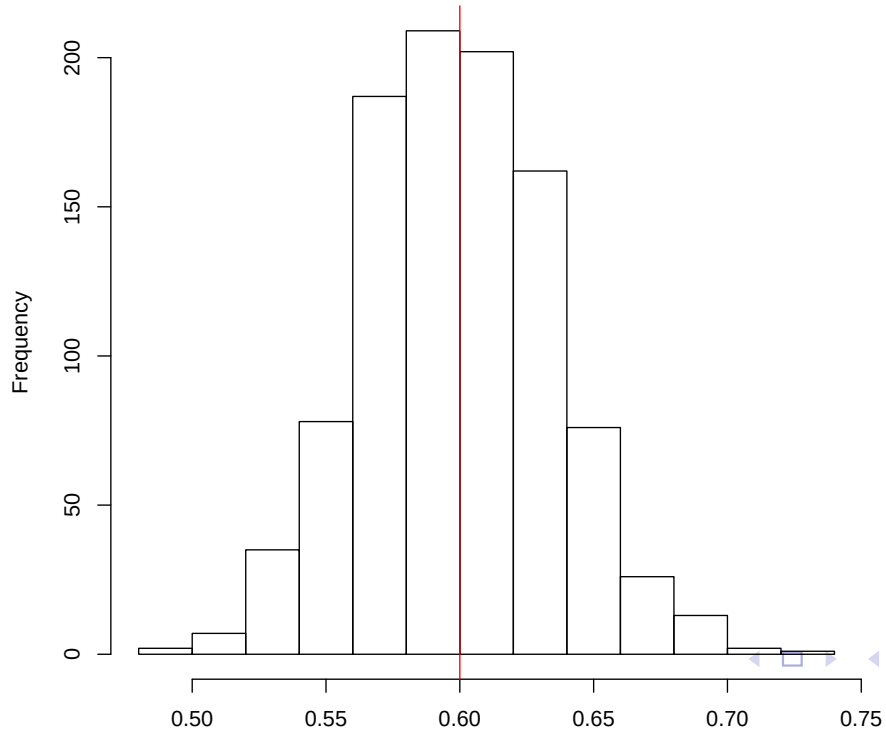
which we can estimate via

$$\hat{\alpha} = \frac{\hat{\sigma}_Y^2 - \hat{\sigma}_{XY}^2}{\hat{\sigma}_X^2 + \hat{\sigma}_Y^2 - 2\hat{\sigma}_{XY}^2}$$

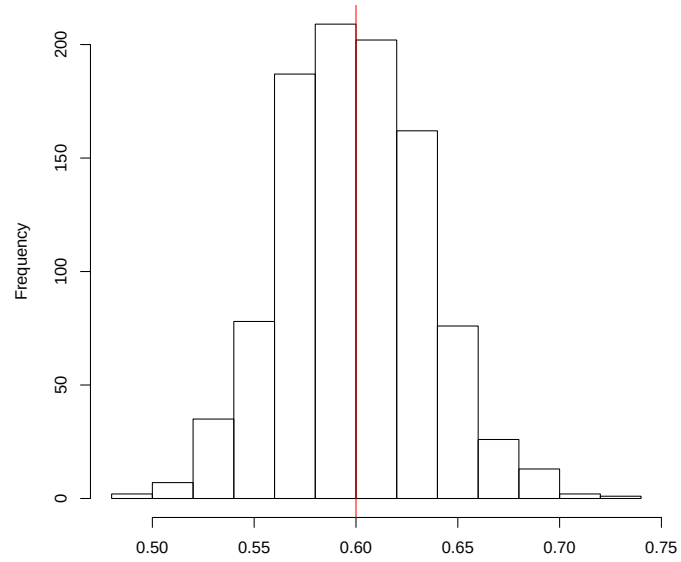
Bootstrap detour

Now that we have an estimator of α, it would be nice to have an estimator of its variability. In this case, computing a standard error is very difficult.

Suppose for a moment that we can simulate a large number of draws (say 1000) of the data, which has actual value $\alpha = 0.6$. Then we could get estimates $\hat{\alpha}_1, \ldots, \hat{\alpha}_{1000}$.



Bootstrap detour



This is a histogram of all 1000 estimates. The mean of all of these is:

$$\overline{\alpha} = \frac{1}{1000}\sum_{r=1}^{1000}\hat{\alpha}_r = 0.599,$$

which is very close to 0.6 (red line), and the standard error is

$$\sqrt{\frac{1}{1000-1}\sum_{r=1}^{1000}(\hat{\alpha}_r - \overline{\alpha})^2} = 0.035.$$

Bootstrap detour

The standard error of 0.035 gives a very good idea of the accuracy of $\hat{\alpha}$ for a single sample. Roughly speaking, for a new random sample, we expect $\hat{\alpha} \in (\alpha - 2 * 0.035, \alpha + 2 * 0.035)$.

In practice, of course, we cannot use this procedure as it relies on being able to draw a large number of (independent) samples from the same distribution as our data.

This is where the bootstrap comes in.

We instead draw a large number of samples directly from our observed data. This sampling is done with replacement, which means that the same data point can be drawn multiple times.

Bootstrap detour: Small example

Suppose we have data $X = (4.3, 3, 7.2, 6.9, 5.5)$.

Then we can draw bootstrap samples, which might look like:

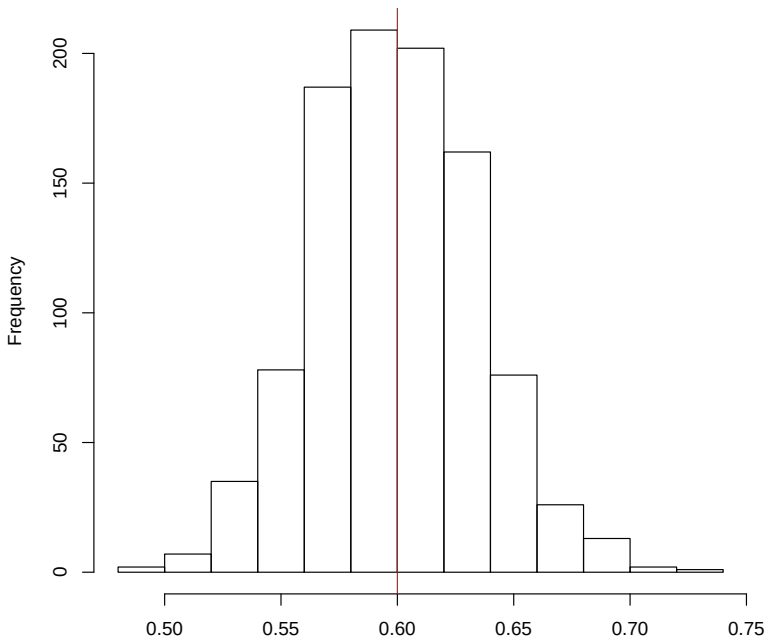
$$
\begin{aligned}
X_1^* &= (7.2, 4.3, 7.2, 5.5, 6.9) \\
X_2^* &= (6.9, 4.3, 3.0, 4.3, 6.9) \\
&\vdots \\
X_B^* &= (4.3, 3.0, 3.0, 5.5, 6.9)
\end{aligned}
$$

It turns out these are all draws from the same (empirical) distribution as our data, which should be close to the actual distribution (Glivenko-Cantelli theorem).

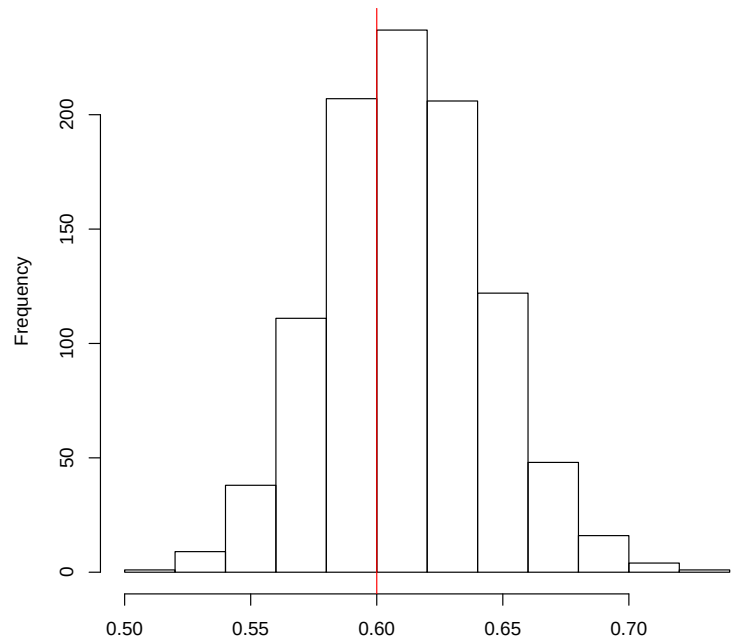
Bootstrap detour

Now, the bootstrap estimator of the standard error is:

$$\mathrm{SE}_B(\hat{\alpha}) = \sqrt{\frac{1}{B-1}\sum_{b=1}^{B}\left(\hat{\alpha}_b* - \frac{1}{B}\sum_{s=1}^{B}\hat{\alpha}_s^*\right)^2}$$



Sampling distribution of $\hat{\alpha}$ from repeated draws of same data (impossible)



Sampling distribution of $\hat{\alpha}$ estimated from bootstrap draw (possible)

Bootstrap: End detour

Summary:

Suppose we have data $Z_1, \ldots, Z_n$ and we want to get an idea of the sampling distribution of some statistic $\hat{f}(Z_1, \ldots, Z_n)$. Then we do the following

1. Choose some large number of samples, B.
2. For each $b = 1, \ldots, B$, draw a new dataset from $Z_1, \ldots, Z_n$, call it $Z_1^*, \ldots, Z_n^*$.
3. Compute $\hat{f}_b^* = \hat{f}(Z_1^*, \ldots, Z_n^*)$.
4. Now, we can estimate the distribution of $\hat{f}(Z_1, \ldots, Z_n)$ by looking at the B draws, $\hat{f}_b^*$.

End detour

Bagging: The bootstrap part

Now, instead of having B separate training sets, we do B bootstrap samples. $\hat{f}_1^*(\mathbf{x}), \ldots, \hat{f}_B^*(\mathbf{x})$, and then average them:

$$\hat{f}_{\text{bag}}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b^*(\mathbf{x})$$

This process is known as Bagging

Note that if we use a linear smoother, then we will reproduce the original estimator as $B \to \infty$

Bagging with squared error

Suppose $Z_i \sim \mathbb{P}$ and $f^*_{bag}(x) = \mathbb{E}\hat{f}^*(x)$, where $\hat{f}^*$ is trained on a new draw[1] from $\mathbb{P}$

$$\begin{aligned}\mathbb{P}(Y - \hat{f}^*(x))^2 &= \mathbb{P}(Y - f^*_{bag}(x))^2 + \mathbb{P}(f^*_{bag}(x) - \hat{f}^*(x))^2 \\ &\geq \mathbb{P}(Y - f^*_{bag}(x))^2\end{aligned}$$

Implication: We cannot increase test squared error by bagging.

[1] The only difference from actual bagging is that here we are drawing from the unknown $\mathbb{P}$ instead of $\hat{\mathbb{P}}$. We'll discuss in a moment the implications of this.

Bagging for classification

For classification, there isn't a straightforward additive decomposition of bias and variance

Bagging a

- good classifier can make it better
- bad classifier can make it worse

Intuition: Suppose the true response is always $Y = 1$ at x. Suppose for a particular classifier $\mathbb{P}(\hat{g}(x) = 1) = 0.49$. Then, $\mathbb{P}(\hat{g}(x) \neq Y) = 0.51$. However, for large B, $\mathbb{P}(\hat{g}_{bag}(x) \neq Y) \to 1$

Bagging trees

Bagging trees

The procedure for trees is the following

1. Choose a large number B.
2. For each $b = 1, \ldots, B$, grow an unpruned tree on the b^{th} bootstrap draw from the data.
3. Average all the predictions of the trees together

   (For classification, the amounts to voting with the highest proportion of classifications)

Each tree, since it is unpruned, will have high variance and low bias.

Therefore averaging many trees results in an estimator that has lower variance and still low bias.

Bagging trees

If the class probability estimates are desired, do not use these bootstrap proportions

Intuition: Suppose the true probability of a 1 at x is 0.66. If each of the bagged classifiers are correct, then our estimate is 1.

It is better to average the probability estimates instead (the proportions returned by each tree at a terminal node)

Bagging trees

Now that we are growing a large number (B) of random trees, we can't directly look at the dendrogram

We no longer have that nice diagram that shows the segmentation of the predictor space (More accurately, we have B of them)

However, we do get some helpful information instead:

- Mean decrease variable importance
- Permutation variable importance
- Out-of-Bag error estimation (OOB)

  (Each time a tree is grown, we can get its prediction error on the unused observations. We average this over all bootstrap samples)
- Proximity plot

Bagging trees: Mean decrease

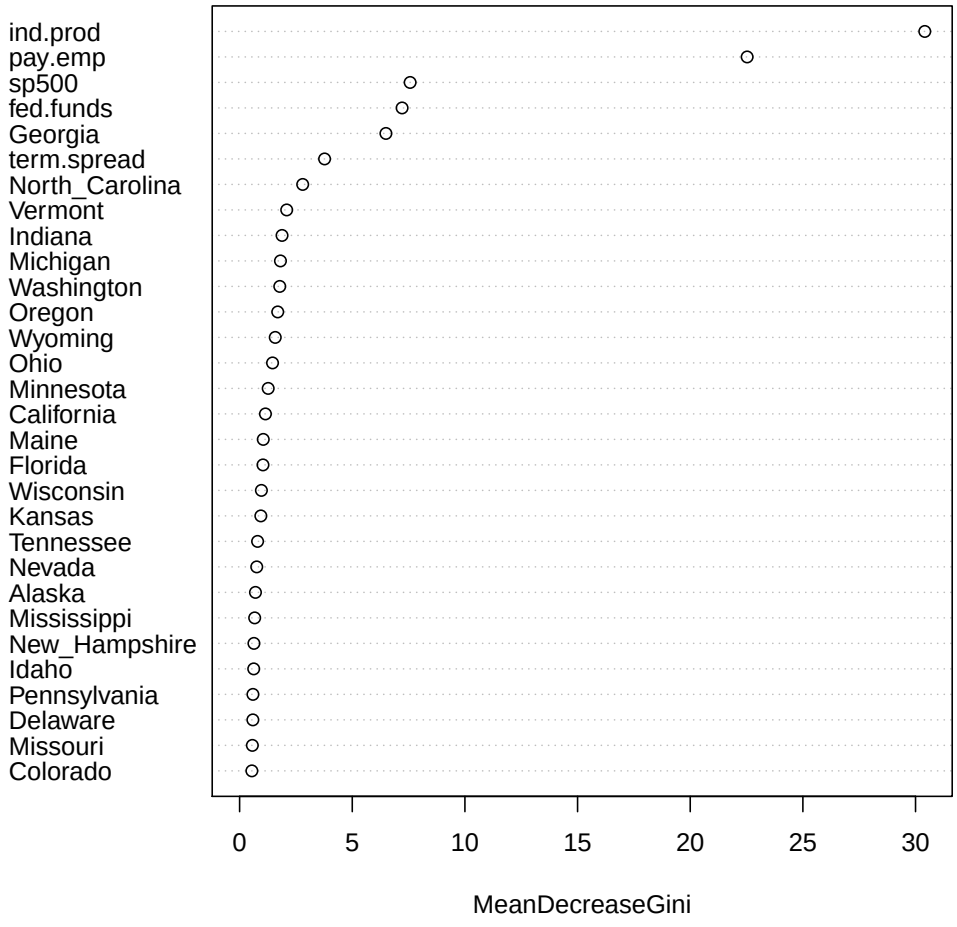
To recover some information, we can do the following:

1. For each of the B trees and each of the p variables, we record the amount that the Gini index is reduced by the addition of that variable
2. Report the average reduction over all B trees.

This gives us an indication of the importance of a variable[2]

[2] A very important caveat to this interpretation will be coming up when we talk about adding random, uncorrelated predictors

Bagging trees: Gini importance



Out-of-Bag samples (OOB)

One can show that, on average, drawing n samples from n observations with replacement results in about $2/3$ of the observations being selected.

The remaining one-third of the observations not used are referred to as out-of-bag (OOB)

We can think of it as a for-free cross-validation

Bagging trees: Permutation

Consider the b^{th} tree T_b

1. The OOB prediction accuracy of T_b is recorded
2. Then, the j^{th} variable is randomly permuted in the OOB samples (ie: If $\mathcal{D}_{OOB,b} = (Z_{t_1}, \ldots, Z_{t_{n/3}})$ then draw a new $t_1^*, \ldots, t_{n/3}^*$ without replacement and predict $\mathcal{D}^*_{OOB,b} = (Z^*_{t_1}, \ldots, Z^*_{t_{n/3}})$, where $Z^*_{t_i} = (Y_{t_i}, X_{t_i,1}, \ldots, X_{t_i^*,j}, \ldots, X_{t_i,p}))$
3. The prediction error is recomputed and the change in prediction error is recorded

Proximity plot

For the b^{th} tree, we can examine which OOB observations are assigned to the same terminal node

Form an $n \times n$ matrix P and increment $P[i, i'] \leftarrow P[i, i'] + 1$ if Z_i and $Z_{i'}$ are assigned to the same terminal node

Now, use some sort of dimension reduction technique to visualize the data in 2-3 dimensions

(Multidimensional scaling is most commonly used (between observation distances are preserved))

The idea is that even if the data have combinations of qualitative/quantitative variables and/or have high dimension, we can view their similarity through the forward operator of the bagged estimator

Random Forest

Random Forest

Random Forest is a small extension of Bagging, in which the bootstrap trees are decorrelated

The idea is, we draw a bootstrap sample and start to build a tree.

- At each split, we randomly select m of the possible p predictors as candidates for the split.
- A new sample of size m of the predictors is taken at each split.

Usually, we use about $m = \sqrt{p}$ ($p/3$ for regression)$^\sharp$

(This would be 7 out of 56 predictors for GDP data)

In other words, at each split, we aren't even allowed to consider the majority of possible predictors!

Random forest

What is going on here?

Suppose there is 1 really strong predictor and many mediocre ones.

- Then each tree will have this one predictor in it,
- Therefore, each tree will look very similar (i.e. highly correlated).
- Averaging highly correlated things leads to much less variance reduction than if they were uncorrelated.

If we don't allow some trees/splits to use this important variable, each of the trees will be much less similar and hence much less correlated.

Bagging is Random Forest when $m = p$, that is, when we can consider all the variables at each split.

Random forest

An average of B i.i.d random variables has variance

$$\frac{\sigma^2}{B}$$

An average of B i.d random variables has variance (not independent)$^\sharp$

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

for correlation ρ

As $B \to \infty$, the second term goes to zero, but the first term remains

Hence, correlation of the trees limits the benefit of averaging

Sensitivity and specificity for recessions

Sensitivity: The proportion of times we label recession, given that recession is the correct answer.

Specificity: The proportion of times we label no recession, given that no recession is the correct answer.

We can think of this in terms of hypothesis testing. If

$$H_0: \text{ no recession},$$

then

Sensitivity: $P(\text{reject } H_0 | H_0 \text{ is false})$, that is: 1 - P(Type II error)

Specificity: $P(\text{accept } H_0 | H_0 \text{ is true})$, that is: 1 - P(Type I error)

Confusion matrix

Alternatively, we can report our results in a matrix:

		Truth	
		Growth	Recession
Our Predictions	Growth	(A)	(B)
	Recession	(C)	(D)

For each observation in the test set, we compare our prediction to the truth.

The total number of each combination is recorded in the table.

The overall miss-classification rate is

$$\frac{(\mathrm{B}) + (\mathrm{C})}{(\mathrm{A}) + (\mathrm{B}) + (\mathrm{C}) + (\mathrm{D})} = \frac{(\mathrm{B}) + (\mathrm{C})}{n_{\text{test}}}$$

Tree results: Confusion matrices

			Truth		
			Growth	Recession	Mis-Class
Our Preds.	Null	Growth	111	26	
		Recession	0	0	18.9%
	Tree	Growth	99	3	
		Recession	12	23	10.9%
	Random Forest	Growth	102	5	
		Recession	9	21	10.2%
	Bagging	Growth	104	3	
		Recession	7	23	7.3%

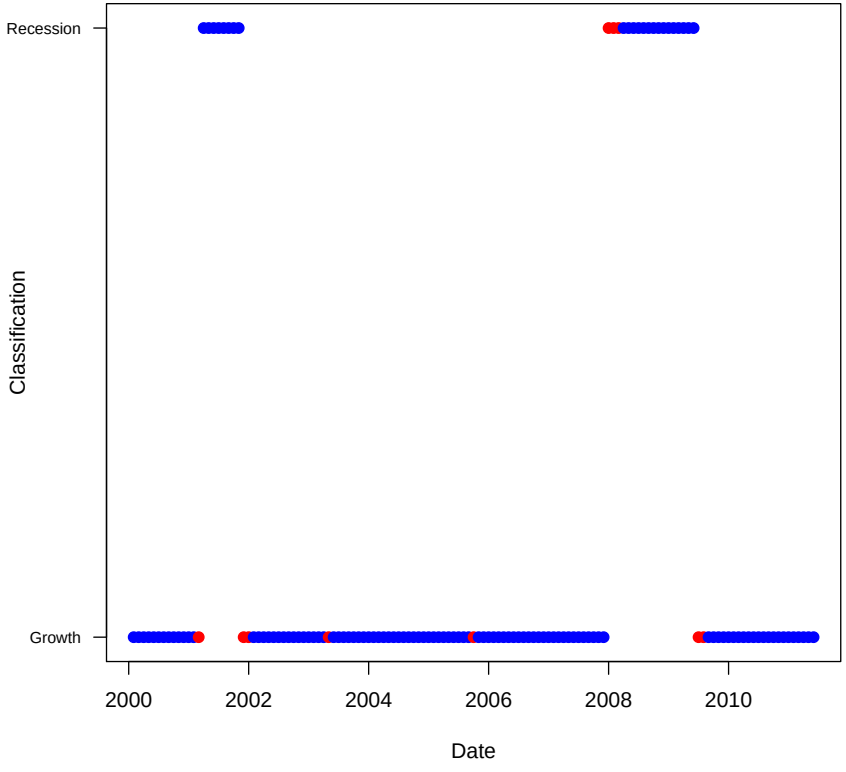
Note that the OOB error estimate for bagging is:

			Truth		
			Growth	Recession	Miss-Class
Our Preds.	Bagging	Growth	400	10	
		Recession	21	46	6.5%

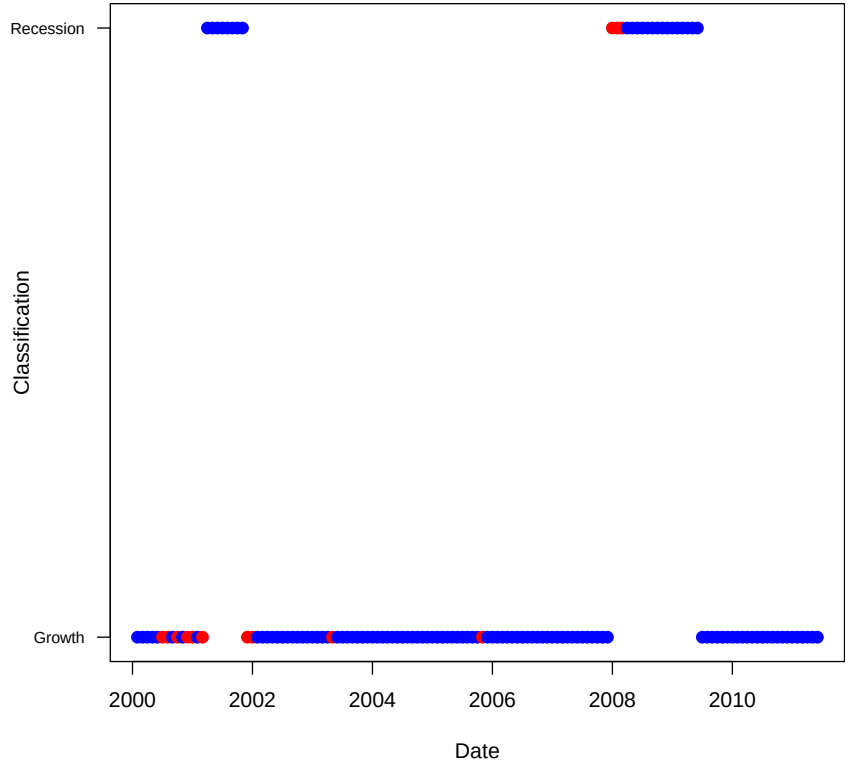
Tree results: Sensitivity & specificity

	Sensitivity	Specificity
Null	0.000	1.000
Tree	0.884	0.891
Random Forest	0.807	0.918
Bagging	0.884	0.936

Results of Bagging on recession data



Bagging



Single tree

Trees in R

```
out.tree   = tree(Y~.,data=X,subset=train,split='dev')
out.tree.cv = cv.tree(out.tree,K=3)
best.k     = out.tree.cv$k[which.min(out.tree.cv$dev)]
out.tree   = prune.tree(out.tree.orig,k=best.k)
#Plots
plot(out.tree.orig)
text(out.tree.orig)
#Bagging/Random Forests
require(randomForest)
out.rf  = randomForest(Y~.,data=X,subset=train,mtry=m)
#Variable importance
varImpPlot(out.rf,type=2,main='')
```

Additional random forest topics

Claim: Random forest cannot overfit. This is and isn't true.

Write

$$\hat{f}_{rf}^{B} = \frac{1}{B}\sum_{b=1}^{B} T(x;\Theta_b)$$

where Θ_b characterizes the b^{th} tree

(That is, the split variables, cutpoints of each node, terminal node values)

Increasing B does not cause Random forest to overfit, rather removes the Monte-Carlo-like approximation error

$$\hat{f}_{rf}(x) = \mathbb{E}_{\Theta} T(x,\Theta) = \lim_{B\to\infty} \hat{f}_{rf}^{B}$$

However, this limit can overfit the data, the average of fully grown trees can result in too complex of a model

(Note that Segal (2004) shows that a small benefit can be derived by stopping each tree short, but thus induce another tuning parameter)

Additional random forest topics

Things I'd like to cover but may not

(AKA possible short presentation topics)

- Variance and decorrelation effect (showing precisely how random forest may work/not work)
- Introduction of noise covariates improves performance
- Using subsampling instead of bootstrap to generate trees

  (This is an idea I had while writing this up. I don't know if this exists, but it seems to work in many cases the bootstrap doesn't and is easier to do theory (just use Hoeffding's inequalty for U-statistics))
- Adaptive nearest neighbors

Boosting

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[3]

- 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 $\mathcal{X}$

[3] 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):

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

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.

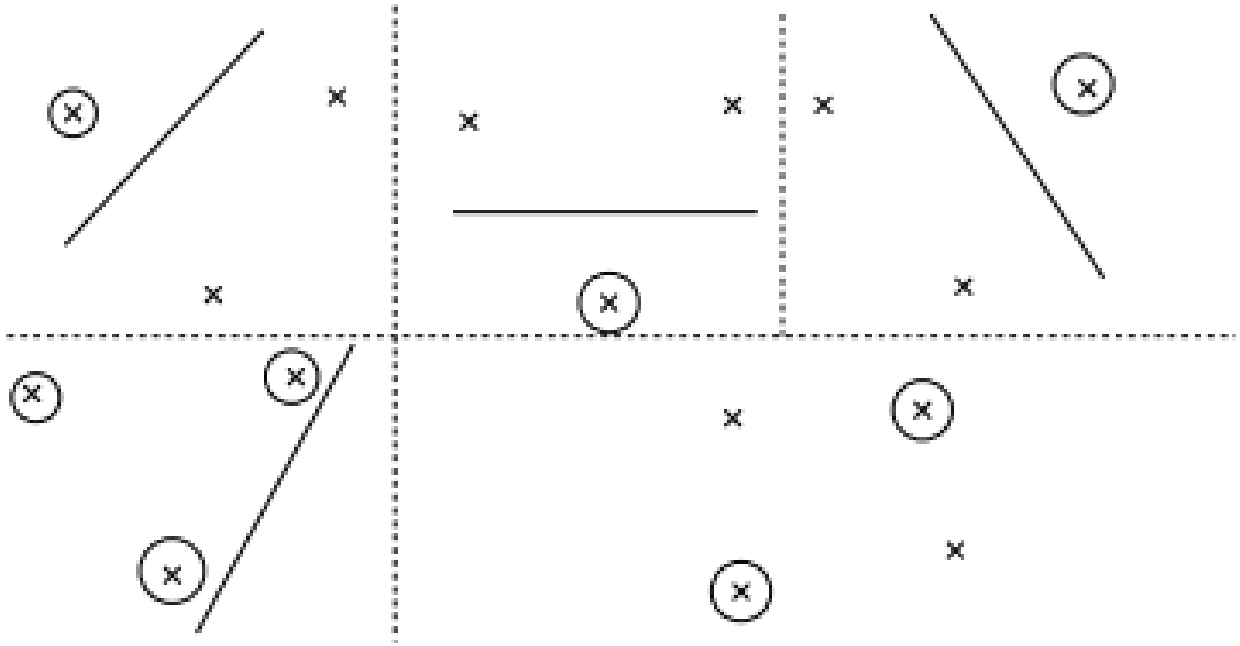
(This should be compared with SVMs, where we wish to separate points with hyperplanes)

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 $F \in \mathcal{F}_4 \Rightarrow d = 3$

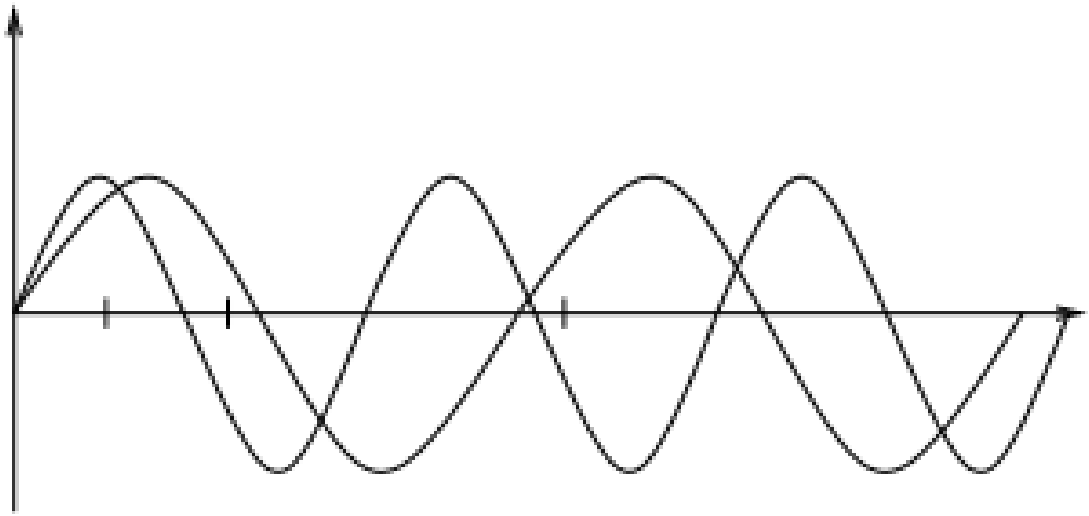


VC-dimension

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[4]

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

[4] 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

Motivation

Reminder: We are attempting to make a classifier $\hat{g}$ such that

$$R(\hat{g}) = \mathbb{P}(\hat{g}(X) \neq Y|\mathcal{D}) \approx \inf_{g} R(g) = \mathbb{P}(\min\{\eta(X), 1-\eta(X)\})$$

(The Bayes' risk)
where $\eta(x) = \mathbb{P}(Y = 1|X = x)$

The infimum is achieved by $g^*(x) = g(2\eta(x) - 1)$, where

$$g(x) = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x \leq 0 \end{cases}$$

Boosting seeks to generate a linear combination of base classifiers

Motivation

Create a committee of classifiers that combines many weak classifiers

Letting

$$\hat{R}(g) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{Y_i \neq g(x)}(X_i)$$

with prediction risk

$$R(g) = \mathbb{E}_Z \mathbf{1}(Y \neq g(X))$$

A weak classifier is a g such that $R(g)$ is only slightly better than random guessing

(These are the weak learners)

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 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 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)

AdaBoost intuition

Often, $\phi_m(x) = \phi(x; \theta_m)$

We'd like to find

$$\min_{(\beta_m),(\theta_m)} \hat{\mathbb{P}}\ell_{(\beta_m),(\theta_m)}$$

For most loss function (ℓ) and basis function ϕ combinations, this is a computationally intensive optimization

This speaks to needing a numerical approximation

AdaBoost intuition

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 revist the fit from any previous step)

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

$$(\beta_m, \theta_m) = \underset{\beta,\theta}{\operatorname{argmin}} \sum_{i=1}^{n} \ell(Y_i, f_{m-1}(X_i) + \beta b(X_i, \theta))$$

(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.)

AdaBoost intuition

However, squared error loss isn't right for classification

AdaBoost implicitly uses the loss function

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

and basis functions g_m

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

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

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

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

Hence, we can find

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

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