Bootstrapping and Bagging

Rebecca C. Steorts

Department of Statistical Sciences
Duke University

Predictive Modeling

November 15, 2015

Here, the bootstrap can be calculated to improve statistical learning of decision trees.

- The key to dealing with uncertainty in parameters and functionals is the sampling distribution of estimators.
- Knowing what distribution we'd get for our estimates on repeating the experiment would give us things like standard errors.

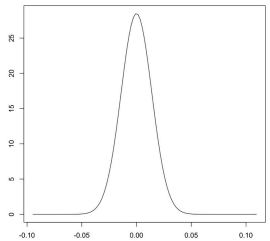
- Efron's insight was that we can *simulate* replication.
- After all, we have already fitted a model to the data, which is a guess at the mechanism which generated the data.
- Running that mechanism generates simulated data which, by hypothesis, has the same distribution as the real data.
- Feeding the simulated data through our estimator gives us one draw from the sampling distribution; repeating this many times yields the sampling distribution.

Since we are using the model to give us its own uncertainty, Efron called this "bootstrapping"

data

.00168
-0.00249
0.0183
-0.00587
0.0139

estimator



fitted model

parameter calculation

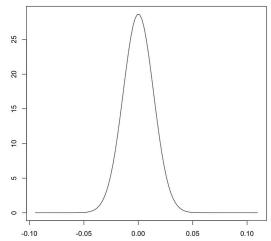
$q_{0.01} = -0.0326$

simulation

simulated data

.00183
-0.00378
0.00754
-0.00587
-0.00673

estimator



re-estimate

$q_{0.01} = -0.0323$

- Suppose the original data is x.
- Our parameter estimate from the data is $\hat{\theta}$.
- Surrogate data sets simulated from fitted model: $\tilde{X}_1, \tilde{X}_2, \ldots \tilde{X}_B$.
- Corresponding re-estimates of the parameters on the surrogate data are $\tilde{\theta}_1, \tilde{\theta}_2, \ldots \tilde{\theta}_B$.
  - Function of interest is estimated by the statistic T, with sample value $\hat{t} = T(x)$,
  - Values of the surrogates of

$$\tilde{t}_1 = T(\tilde{X}_1), \tilde{t}_2 = T(\tilde{X}_2), \ldots \tilde{t}_B = T(\tilde{X}_B).$$

- When the function of interest *is* the parameter everything applies without modification.

We will assume that the model is correct for *some* value of θ, which we will call θ_0. The true (population or ensemble) values of the functional is likewise t_0.

- Simple approach: get the variance or standard error:

$$\widehat{\mathrm{Var}}\left[\hat{t}\right] = \mathrm{Var}\left[\tilde{t}\right] \quad (1)$$

$$\widehat{\mathrm{se}}(\hat{t}) = \mathrm{sd}(\tilde{t}) \quad (2)$$

- Idea: simulated $\tilde{X}$ has about the same distribution as the real X that our data, x, was drawn from.
  - applying the same estimation procedure to the surrogate data gives us the sampling distribution.
- This assumes, of course, that our model is right, and that $\hat{\theta}$ is not too far from θ_0.

Pseudo-code is provided in below:

```
rboot <- function(B, statistic, simulator) {
  tboots <- replicate(B, statistic(simulator()))
  return(tboots)

bootstrap.se <- function(simulator, statistic, B) {
  tboots <- rboot(B, statistic, simulator)
  se <- sd(tboots)
  return(se)
```

- Sketch of code for calculating bootstrap standard errors.
- The function `rboot` generates `B` bootstrap samples (using the `simulator` function) and calculates the statistic g on them (using `statistic`).
- `simulator` needs to be a function which returns a surrogate data set in a form suitable for `statistic`. (How would you modify the code to pass arguments to `simulator` and/or `statistic`?)
- Every use of bootstrapping is going to need to do this, it makes sense to break it out as a separate function, rather than writing the same code many times.
- `bootstrap.se` just calls `rboot` and takes a standard deviation.

Bootstrapping corrects a biased estimator. Sampling distribution of $\tilde{t}$ is close to that of $\widehat{t}$, and $\widehat{t}$ itself is close to t_0, implies

$$\mathbf{E}\left[\widehat{t}\right] - t_0 \approx \mathbf{E}\left[\tilde{t}\right] - \widehat{t} \tag{3}$$

- LHS is the bias that we want to know, and the RHS is what we can calculate with the bootstrap.
- Eq. 3 remains valid so long as the sampling distribution of $\widehat{t} - t_0$ is close to that of $\tilde{t} - \widehat{t}$.
- Weaker requirement than asking for $\widehat{t}$ and $\tilde{t}$ themselves to have similar distributions, or asking for $\widehat{t}$ to be close to t_0.

```
bootstrap.bias <- function(simulator, statistic, B,
  t.hat) {
  tboots <- rboot(B, statistic, simulator)
  bias <- mean(tboots) - t.hat
  return(bias)
}
```

- Sketch of code for bootstrap bias correction.
- Arguments are as in
- Note that `t.hat` is the estimate on the original data.

The Pareto distribution[1], or power-law distribution, is a popular model for data with "heavy tails", i.e. where the probability density $f(x)$ goes to zero only very slowly as $x \to \infty$. The probability density is

$$f(x) = \frac{\theta - 1}{x_0} \left(\frac{x}{x_0} \right)^{-\theta} \tag{4}$$

where x_0 is the minimum scale of the distribution, and θ is the **scaling exponent**. (EXERCISE: show that x_0 is the mode of the distribution.) The Pareto is highly right-skewed, with the mean being much larger than the median.

[1] Named after Vilfredo Pareto, the highly influential late-19th/early-20th century economist, political scientist, and proto-Fascist.

If we know x_0, can show that the maximum likelihood estimator of the exponent θ is

$$\hat{\theta} = 1 + \frac{n}{\sum_{i=1}^{n} \log \frac{x_i}{x_0}} \tag{5}$$

and that this is consistent[2], and efficient.

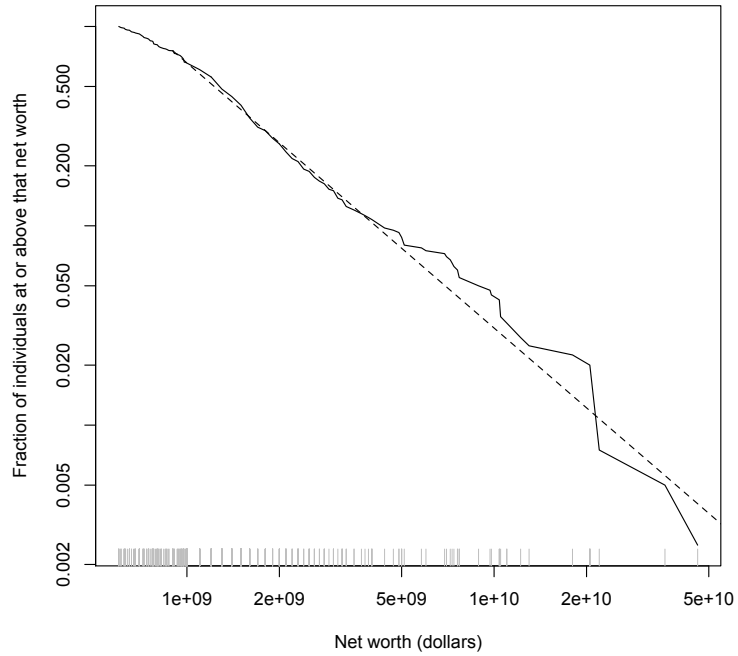
- Picking x_0 is a harder problem — for the present purposes, pretend that the Oracle tells us.
- The file `pareto.R`, on the class website, contains a number of functions related to the Pareto distribution, including a function `pareto.fit` for estimating it. (There's an example of its use below.)

[2] Because the sample mean of $\log X$ converges, under the law of large numbers

- Pareto came up with this density to model the distn of wealth.
- Approximately, but quite robustly across countries and time-periods, the upper tail of the distribution of income and wealth follows a power law, with the exponent varying as money is more or less concentrated among the very richest[3].
- Figure 1 shows the distribution of net worth for the 400 richest Americans in 2003. Taking $x_0 = 9 \times 10^8$, the number of individuals in the tail is 302, and the estimated exponent is $\hat{\theta} = 2.34$.

```
> source("pareto.R")
> wealth <- scan("wealth.dat")
> wealth.pareto <- pareto.fit(wealth,threshold=9e8)
> signif(wealth.pareto$exponent,3)
[1] 2.34
```

[3] Most of the distribution conforms to a log-normal, at least roughly.




```
plot.survival.loglog(wealth,xlab="Net worth (dollars)",
  ylab="Fraction of individuals at or above that net worth")
rug(wealth,side=1,col="grey")
curve((302/400)*ppareto(x,threshold=9e8,exponent=2.34,lower.tail=FALSE),
      add=TRUE,lty=2,from=9e8,to=2*max(wealth))
```

Figure 1: Upper cumulative distribution function (or "survival function") of net worth for the 400 richest individuals in the US (2000 data).

- The solid line shows the fraction of the 400 individuals whose net worth W equaled or exceeded a given value w, $\Pr(W \geq w)$. (Note the logarithmic scale for both axes.)
- The dashed line is a maximum-likelihood estimate of the Pareto distribution, taking $x_0 = \$9 \times 10^8$.
- Since there are 302 individuals at or above the threshold, the cumulative distribution function of the Pareto has to be reduced by a factor of (302/400).

How much uncertainty is there in this estimate of the exponent? Let's bootstrap.

- We need a function to generate Pareto-distributed random variables; this, along with some related functions, is part of the file `pareto.R` on the course website.
- With that tool, parametric bootstrapping proceeds shown above.

```
rboot.pareto <- function(B,exponent,x0,n) {
  replicate(B,pareto.fit(rpareto(n,x0,exponent),x0)$exponent)
}

pareto.se <- function(B,exponent,x0,n) {
  return(sd(rboot.pareto(B,exponent,x0,n)))
}

pareto.bias <- function(B,exponent,x0,n) {
  return(mean(rboot.pareto(B,exponent,x0,n)) - exponent)
}
```

- With $\hat{\theta} = 2.34$, $x_0 = 9 \times 10^8$, $n = 302$ and $B = 10^4$, this gives a standard error of ± 0.077.
- This matches some asymptotic theory but didn't require asymptotic assumptions.
- Asymptotically, the bias is known to go to zero; at this size, bootstrapping gives a bias of 3×10^{-3}, which is effectively negligible.

Why on earth do we need to know about bootstrapping?

- The decision trees in the last chapter suffer from high variance,
- If we split the training data into two parts at random and fit a decision tree to both halves, the results could be quite different.
- In contrast, a procedure with low variance will yield similar results if applied repeatedly to distinct data sets;
    - linear regression tends to have low variance, if the ratio of n to p is moderately large.

- Bootstrap aggregation or *bagging* is a general-purpose procedure for reducing the variance of a statistical learning method.
- We use it here for decision trees.
- Suppose n independent observations

$$Z_1, \ldots, Z_n \sim (\bar{Z}, \sigma^2).^4$$

- How to reduce the variance and increase the prediction accuracy?
  - Take *many training sets* from the population.
  - Guild a separate prediction model using each training set.
  - Average the resulting predictions.

[4] So, averaging a set of observations reduces the variance.

We could do the following:

1. Calculate $\hat{f}^1(x), \ldots, \hat{f}^B(x)$ using B separate training sets
2. Average these to find a single low-variance learning model given by

$$\hat{f}_{\text{avg}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^b(x).$$

This is not practical! Why?

We generally do not have access to multiple training sets. However, instead, we can bootstrap. That is, we can take repeated samples from the single training data set.

1. We generate B different bootstrapped training data sets.
2. Then we train our method on the bth bootstrapped training set in order to get $\hat{f}^{b*}(x)$.
3. Finally, we average all the predictions to obtain

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

This is called *bagging*.

Using `Heart` data: Survival of patients on the waiting list for the Stanford heart transplant program.

- Figure 8.8 shows the results from bagging trees.
- The test error rate is shown as a function of B, the number of trees constructed using bootstrapped training data sets.
- We see that the bagging test error rate is slightly lower in this case than the test error rate obtained from a single tree.
- The number of trees B is not a critical parameter with bagging; using a very large value of B will not lead to overfitting.
- In practice we use a value of B sufficiently large that the error has settled down.
- Using $B = 100$ is sufficient to achieve good performance here.

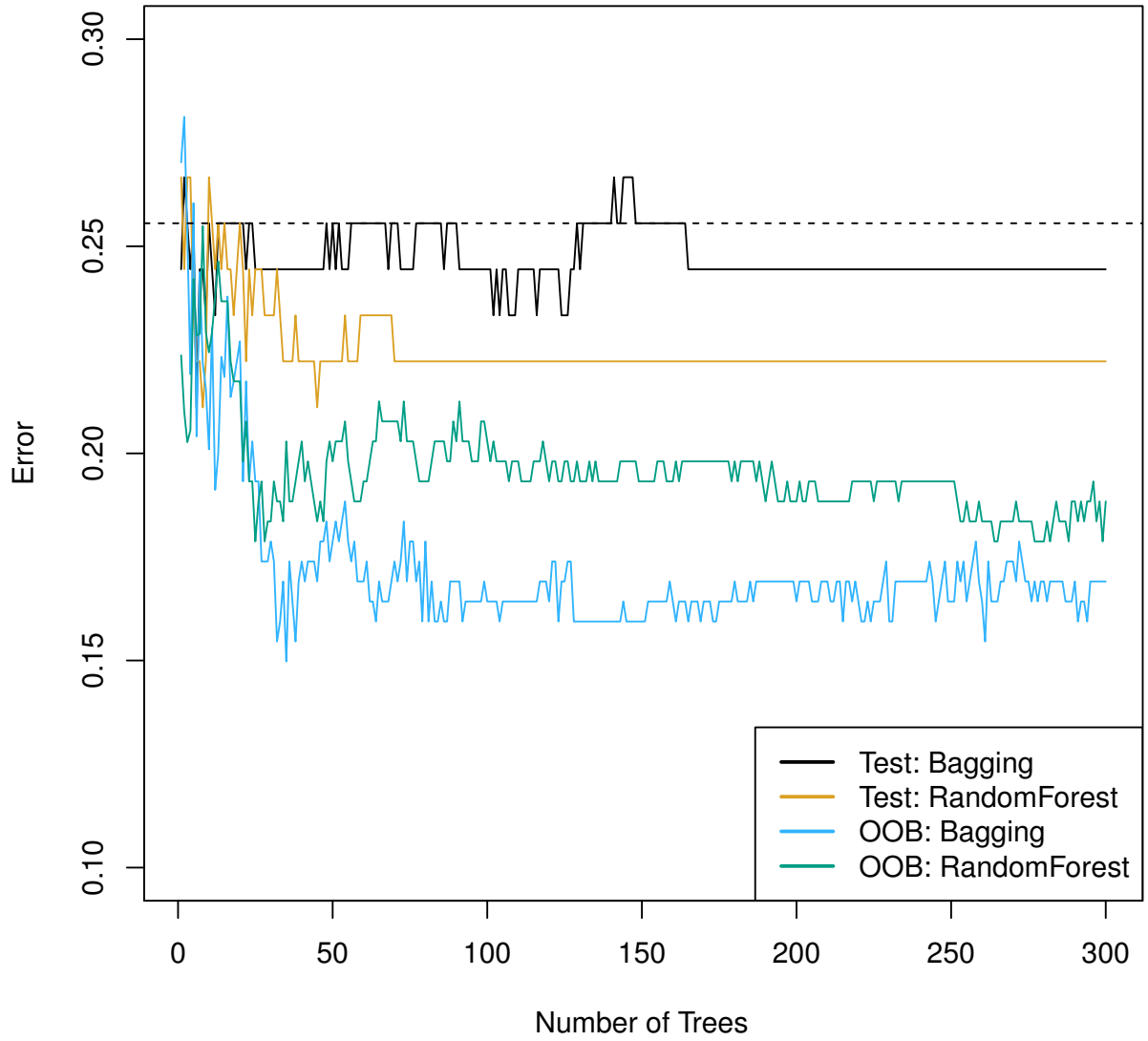


Figure 2: default

Variable importance computed using the mean decrease in Gini index and expressed relative to the maximum.