Classification: Gradient Boosting, Bagging and Random Forests

Siva Balakrishnan
Data Mining: 36-462/36-662

February 21st, 2019

Chapter 8.2 of ISL and Chapter 10 of ESL

Quick questions

- How many of you are comfortable with linear algebra?
  Particularly: eigenvectors, eigenvalues, singular value decomposition.
  (Many things we will see soon-ish – PCA, spectral clustering, matrix completion – will depend heavily on this.)

- Riccardo's office hours.

Recap: Ensemble Classifiers

- Construct many different classifiers $\{\underline{\widehat{f}_1}, \ldots, \underline{\widehat{f}_b}\}$. Combine their predictions by taking a (weighted) majority vote:

$$\widehat{f}(x) = \text{sign}\Big(\sum_{t=1}^{b} \underbrace{\alpha_t}_{\text{weight of } \hat{f}_t .} \underbrace{\widehat{f}_t(x)}_{\text{each } \hat{f}_t \in \{+1,-1\}}\Big),$$

for some weights α_t.

- Why do we call this voting?

$$\sum_{t:\, \hat{f}_t(x) = +1} \alpha_t \overset{?}{\gtrless} \sum_{t:\, \hat{f}_t(x) = -1} \alpha_t .$$

Recap: Boosting Key Ideas

- Want to force each classifier to bring something new (and useful) to the ensemble.

  * weight differently in each iteration.

- How much should we value each new classifier?

  * Roughly, if $\hat{f}_t$ was accurate then higher weight.

4

Recap: The AdaBoost Algorithm

Initialize: Weights $w_i = 1/n$

For $b = 1, \ldots, B$**:**

1. Fit classification tree $\widehat{f}^b$ to the training data with weights $w_1, \ldots, w_n$.
2. Compute weighted misclassification error:

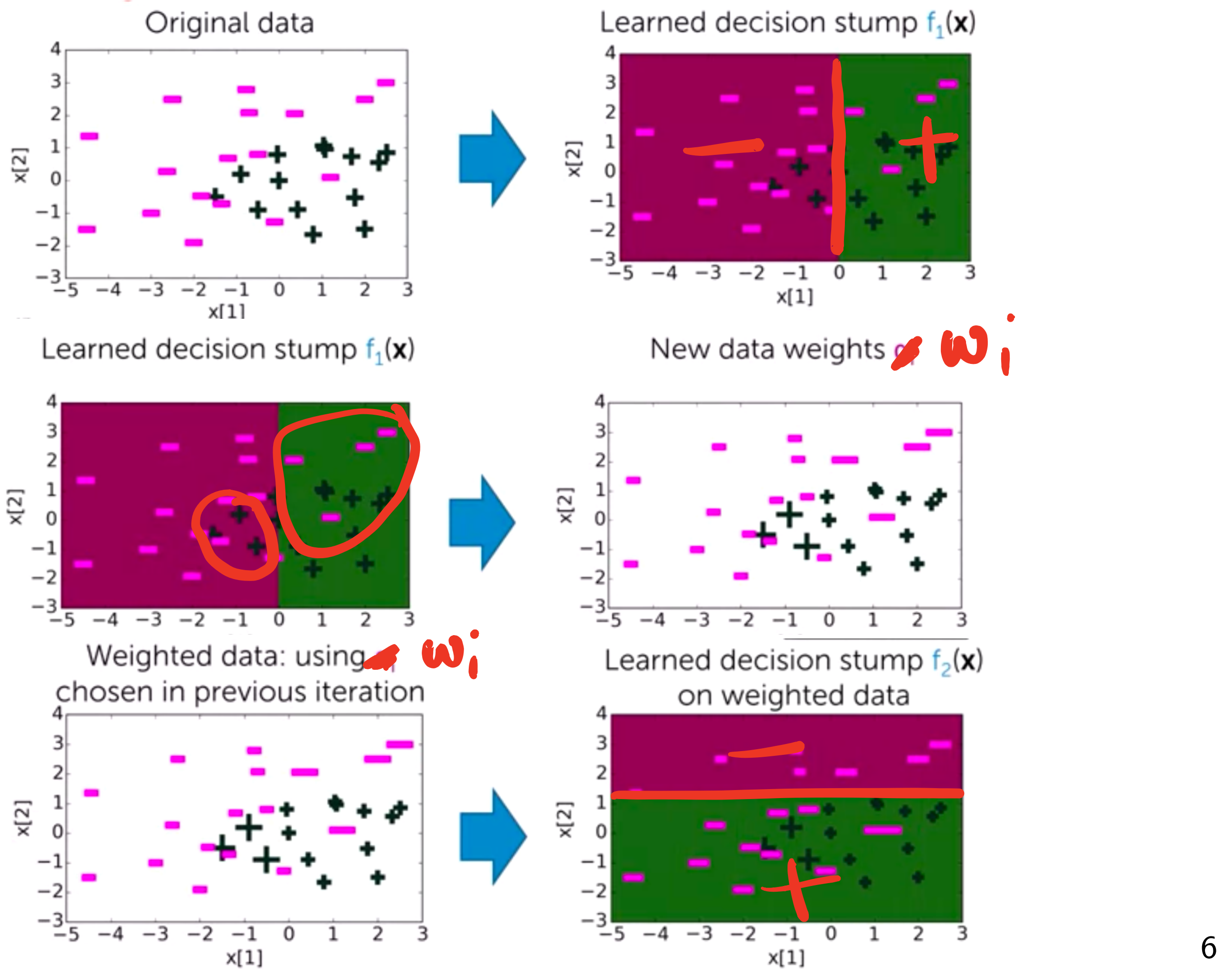
$$\boxed{e_b} = \frac{\sum_{i=1}^n w_i \mathbb{I}\{y_i \neq \widehat{f}^b(x_i)\}}{\sum_{i=1}^n w_i}$$

3. Define $\alpha_b = \log \frac{1-e_b}{e_b}$
4. Update the training data weights:

$$w_i \leftarrow w_i \cdot \exp\left(\alpha_b \mathbb{I}\{y_i \neq \widehat{f}^b(x_i)\}\right)$$

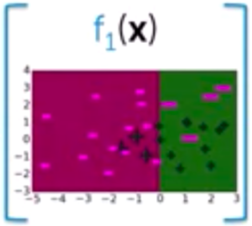
Result: $\widehat{f}(x) = \text{sign}\left(\sum_{b=1}^B \alpha_{\cancel{b}} \underline{\widehat{f}^b(x)}\right)$

Boosting in Pictures (from Carlos Guestrin)

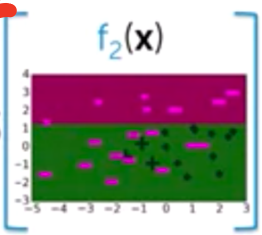


Boosting in Pictures

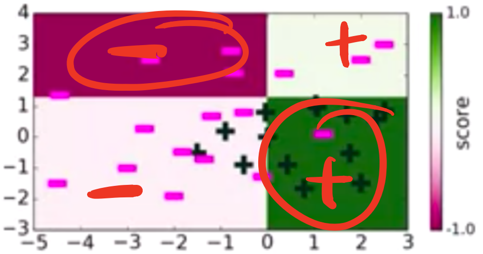
α_1 ~~w_1~~ 0.61

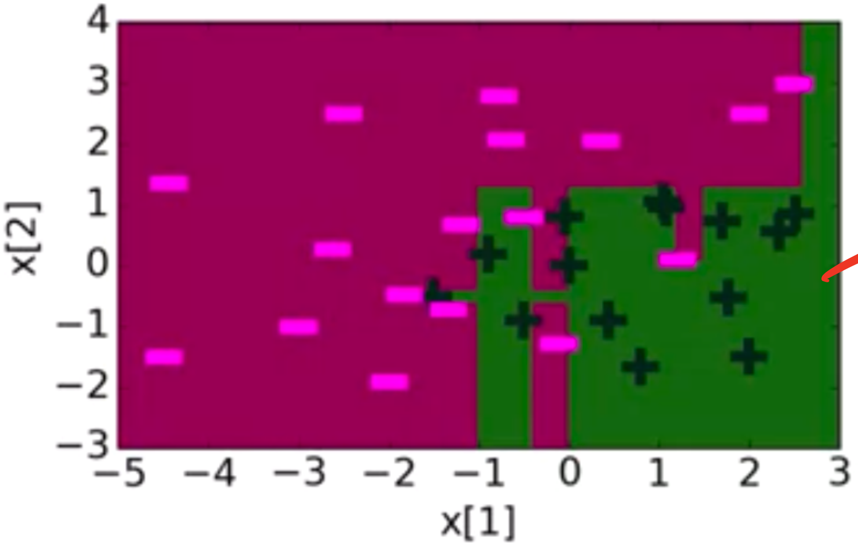


+ α_2 ~~w_2~~ 0.53



=





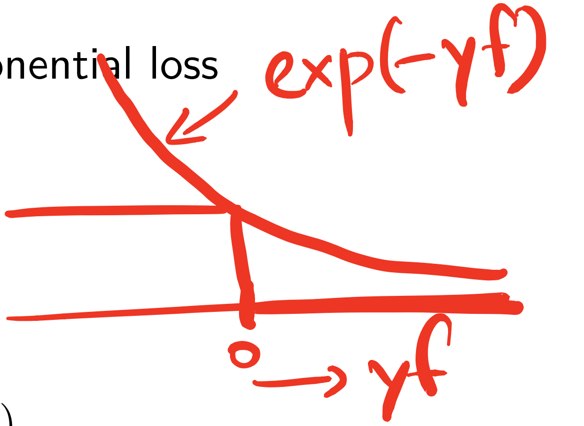
after 30 rounds of boosting

Recap: Boosting the Alternate Viewpoint

Adaboost is just

1. Empirical risk minimization with an exponential loss

$$\frac{1}{n}\sum_{i=1}^{n} e^{-y_i f(x_i)}$$



2. Assuming an additive model of trees:

$$\widehat{f}(x) = \sum_{b=1}^{B} \alpha_b \widehat{f}^{(b)}(x)$$

with $\widehat{f}^{(b)}(x)$ constrained to be a tree.

3. And greedy, stepwise optimization

$$\boxed{\underset{\alpha_b, \widehat{f}^b}{\operatorname{argmin}}} \sum_{i=1}^{n} \exp\left(-y_i \cdot \left(\underbrace{\sum_{k=1}^{b-1} \alpha_k \widehat{f}^k(x)}_{\text{fixed}} + \underbrace{\alpha_b \widehat{f}^b(x)}_{\text{add one new stump}}\right)\right)$$

Can now play with the loss and design "regression boosting" algorithms.

Gradient Boosting for Regression

1. Key idea: Suppose we want to do regression. Fit an ensemble of regression trees, incrementally, i.e.:

$$\widehat{f}(x) = \sum_{b=1}^{B} \alpha_b \widehat{f}^{(b)}(x)$$

   where each $\widehat{f}^{(b)}(x)$ is a regression tree.

2. We simply fit <u>a small regression tree</u> (a stump), find the residual

$$R(x_i) = \underbrace{y_i - \widehat{f}^{(1)}(x_i)}$$

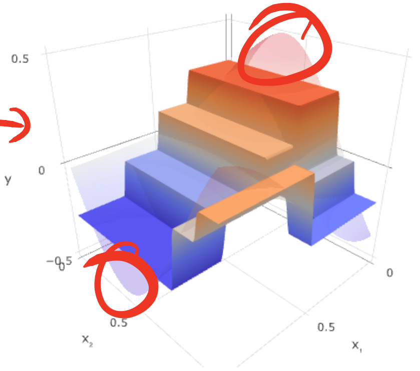
   and now fit a new tree to the residuals, i.e. replace y_i with $R(x_i)$, and repeat.

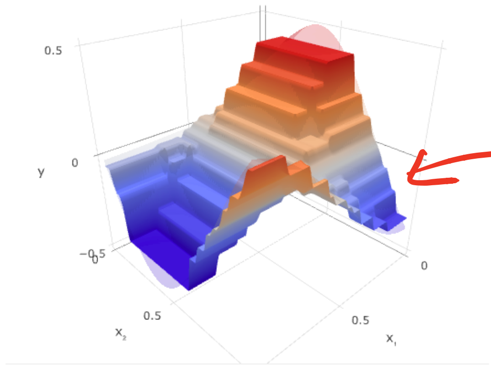
3. In practice, this is a very popular and powerful regression algorithm.

Gradient Boosting in Pictures

- There is a very neat applet that you can play with here: `https://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html`

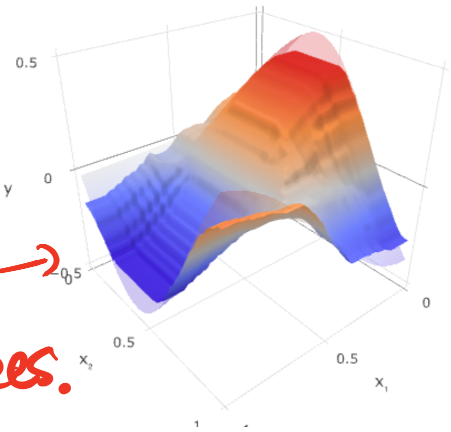
low depth tree

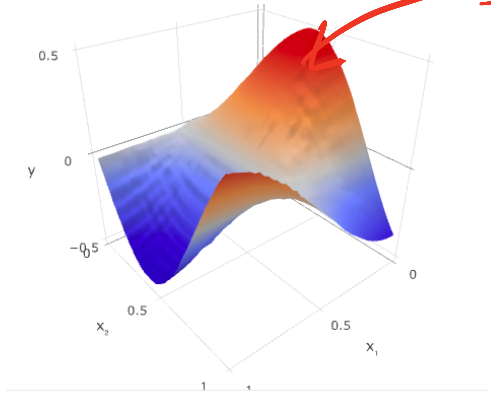




larger tree.

fit 100 depth 3 trees.





fit 100 depth 6 trees.

Boosting discussion

In boosting, we tend to use very simple estimators, like very shallow trees. These have low variance, but are high bias. Because the sequence of trees can adjust for previous errors, they can fix the bias!

The shallower trees can also lead to computational speedups in evaluation, since you don't have to evaluate 500 deep trees (this comment will be clearer at the end of today).

Boosting discussion

Boosting is one of the most powerful classical methods. When well-tuned, it can beat most other classifiers.

However, it requires more tuning than random forests.

Tuning parameters:

- Number of trees, B
- Number of splits in each tree $\leftarrow$
- Other parameters we did not talk about: Amount of subsampling, η
- Other parameters we did not talk about: Amount of shrinkage, ν

In boosting, choosing B too large can in some cases cause overfitting (though miraculously not always). Choosing it too small can give a bad classifier.

B is the main tuning parameter. The others can be tuned more roughly, since it doesn't matter quite as much.

Reducing the variance

We've seen that decision trees are very flexible, but have high variance. This leads to poor classification error.

How can we reduce variance?

average indep. things will reduce variance.

We know that averaging independent things ~~tends~~ to reduce variance. But independent trees would require more data!

increases variance

How could we achieve something similar?

split data, fit tree on each split, average their predictions

reduces variance

The bootstrap

The bootstrap[1] is a fundamental resampling tool in statistics.

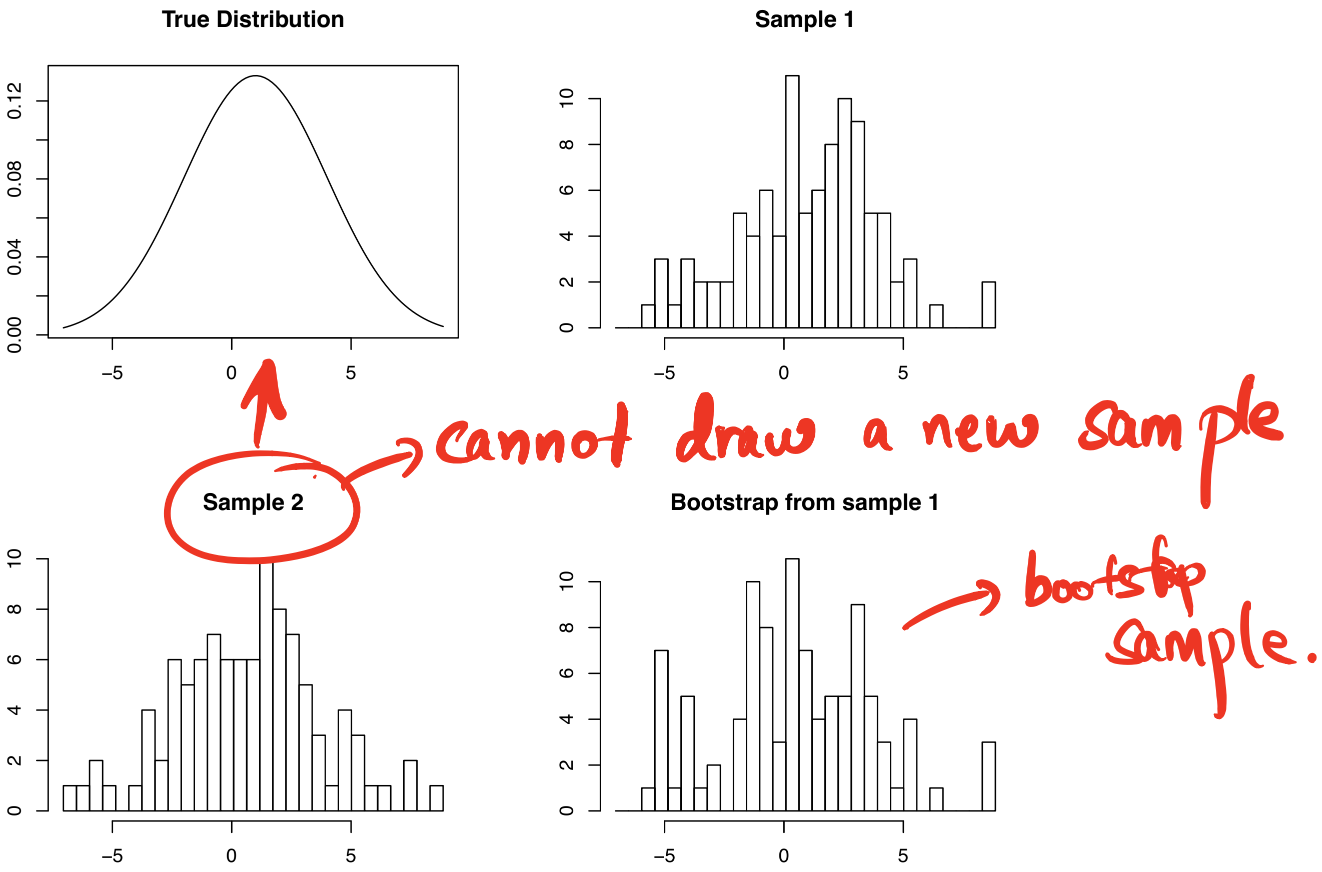
The basic idea underlying the boostrap is that we can estimate the true distribution $\mathcal{F}$ by the so-called empirical distribution $\widehat{\mathcal{F}}$

Given the training data (x_i, y_i), $i = 1, \ldots n$, the empirical distribution function $\widehat{\mathcal{F}}$ is simply

$$P_{\widehat{\mathcal{F}}}\{(X, Y) = (x, y)\} = \begin{cases} \frac{1}{n} & \text{if } (x, y) = (x_i, y_i) \text{ for some } i \\ 0 & \text{otherwise} \end{cases}$$

This is just a discrete probability distribution, putting equal weight $(1/n)$ on each of the observed training points

[1] Efron (1979), "Bootstrap Methods: Another Look at the Jacknife"



With the bootstrap, we are approximating the true distribution by a discrete distribution over the original sample data points.

usually n.

A bootstrap sample of size m from the training data is

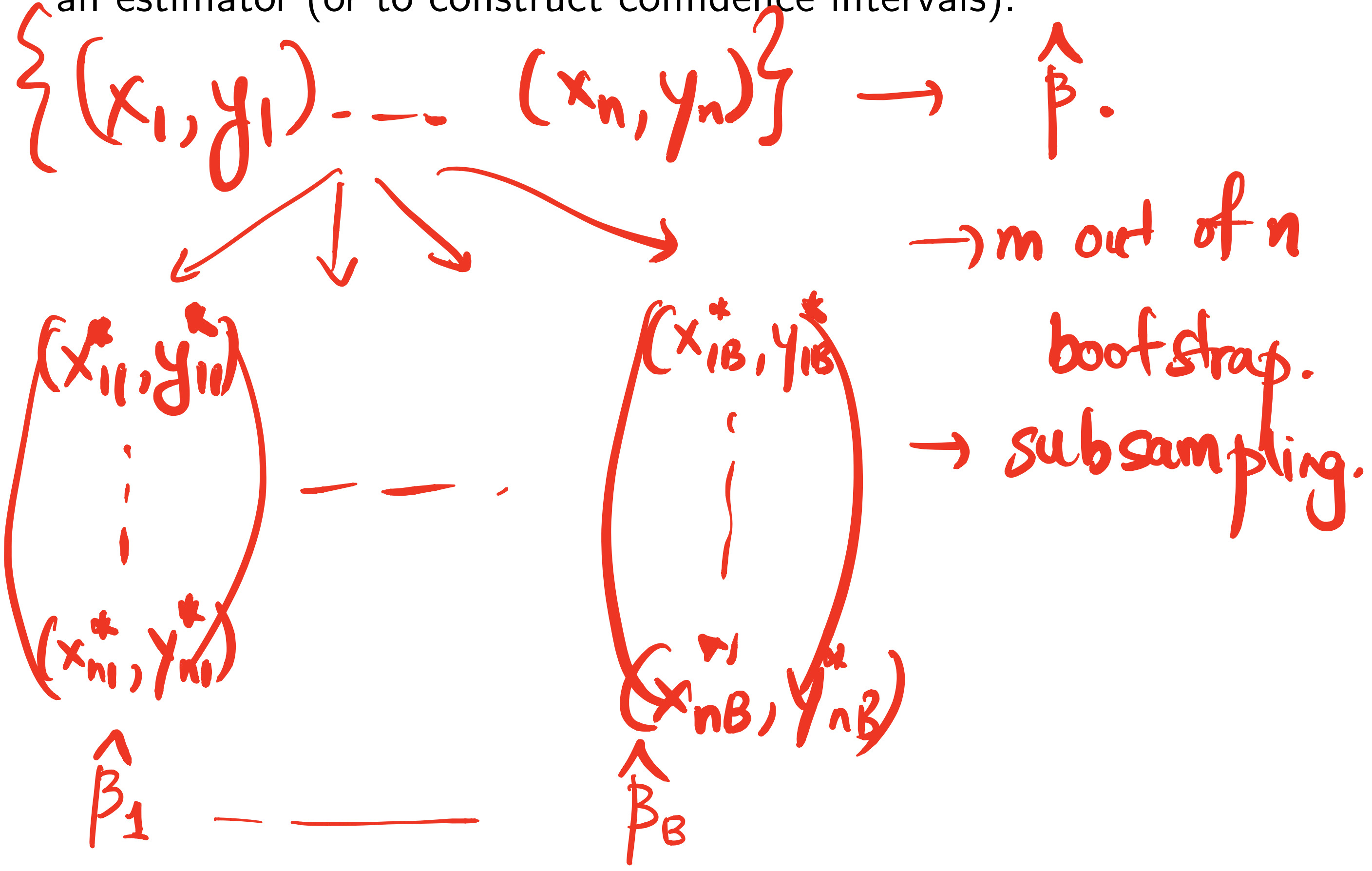
$$(x_i^*, y_i^*),\ i = 1, \ldots m$$

where each (x_i^*, y_i^*) are drawn from uniformly at random from $(x_1, y_1), \ldots (x_n, y_n)$, with replacement

This corresponds exactly to m independent draws from $\widehat{\mathcal{F}}$. Hence it approximates what we would see if we could sample more data from the true $\mathcal{F}$. We often consider $m = n$, which is like sampling an entirely new training set.

Bootstrap: Assessing Variance of an Estimator

- The typical use case of the bootstrap is to assess variance of an estimator (or to construct confidence intervals).



$\{(x_1, y_1) \ldots (x_n, y_n)\} \rightarrow \hat{\beta}$.
$(x^*_{11}, y^*_{11}) \ldots (x^*_{n1}, y^*_{n1})$
$(x^*_{1B}, y^*_{1B}) \ldots (x^*_{nB}, y^*_{nB})$
$\hat{\beta}_1$
$\hat{\beta}_B$
→ m out of n bootstrap.
→ subsampling.

What shows up?

Note: not all of the training points are represented in a bootstrap sample, and some are represented more than once. When $m = n$, about 36.8% of points are left out, for large n (Try to prove this).

These left out points feel almost like a validation set. . . (Later)

Bagging

Bootstrapping gives us an approach to variance reduction!

We are worried that our tree could change dramatically if we drew a slightly different sample.

What if we draw many bootstrap data sets, fit a new tree on each one, and "average" their predictions (somehow)! Now we'll see all the different trees that might show up, and their combination will be stable!

You are likely accustomed to seeing the bootstrap for estimating errors. Now we're using it to construct predictions!

Bagging (more carefully)

Given a training data $\underline{(x_i, y_i)}$, $i = 1, \ldots n$, bagging[2] averages the predictions from predictors (here trees) over a collection of boostrap samples.

For $\underline{b = 1, \ldots B}$ (e.g., $B = 100$), we draw n boostrap samples $\underline{(x_i^{*b}, y_i^{*b})}$, $i = 1, \ldots n$, and we fit a $\underline{\text{classification tree } \hat{f}^{\text{tree},b}}$ on each sampled data set.

↳ less/no pruning.

To classify an input $x \in \mathbb{R}^p$, we simply take the most commonly predicted class:

B votes, take majority

$$\underline{\hat{f}^{\text{bag}}(x) = \underset{k=1,\ldots K}{\text{argmax}} \sum_{b=1}^{B} 1\{\hat{f}^{\text{tree},b}(x) = k\}} \rightarrow \text{less clever boosting.}$$

This is just choosing the class with the most votes.

↳ consensus method

[2] Breiman (1996), "Bagging Predictors"

Tree depth

Most commonly, trees are grown fairly large on each sample, with no pruning. Why are we less worried about tuning each tree?

Because averaging will reduce variance.

What happens to variance as we average many things together?

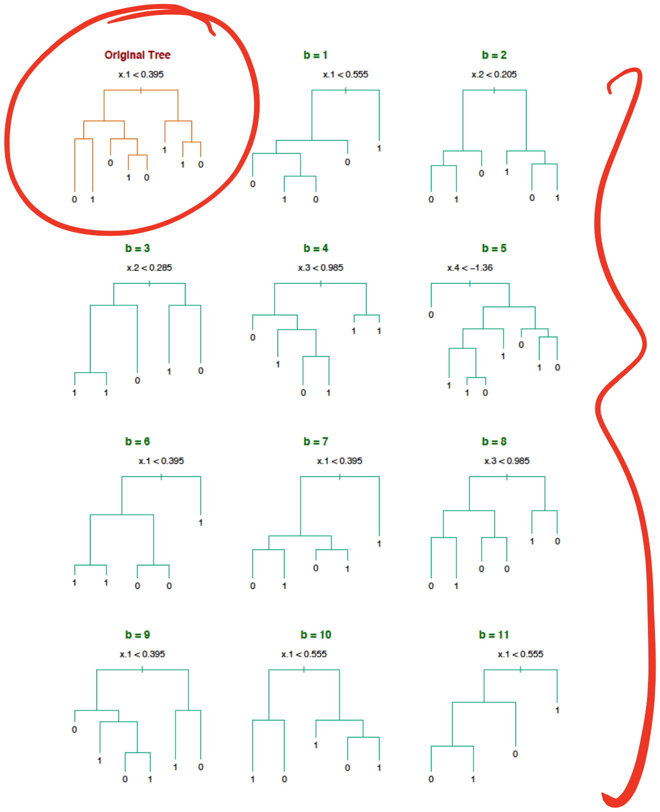
Hopefully go down.

What happens to bias as we average many things together?

Bias won't change much.

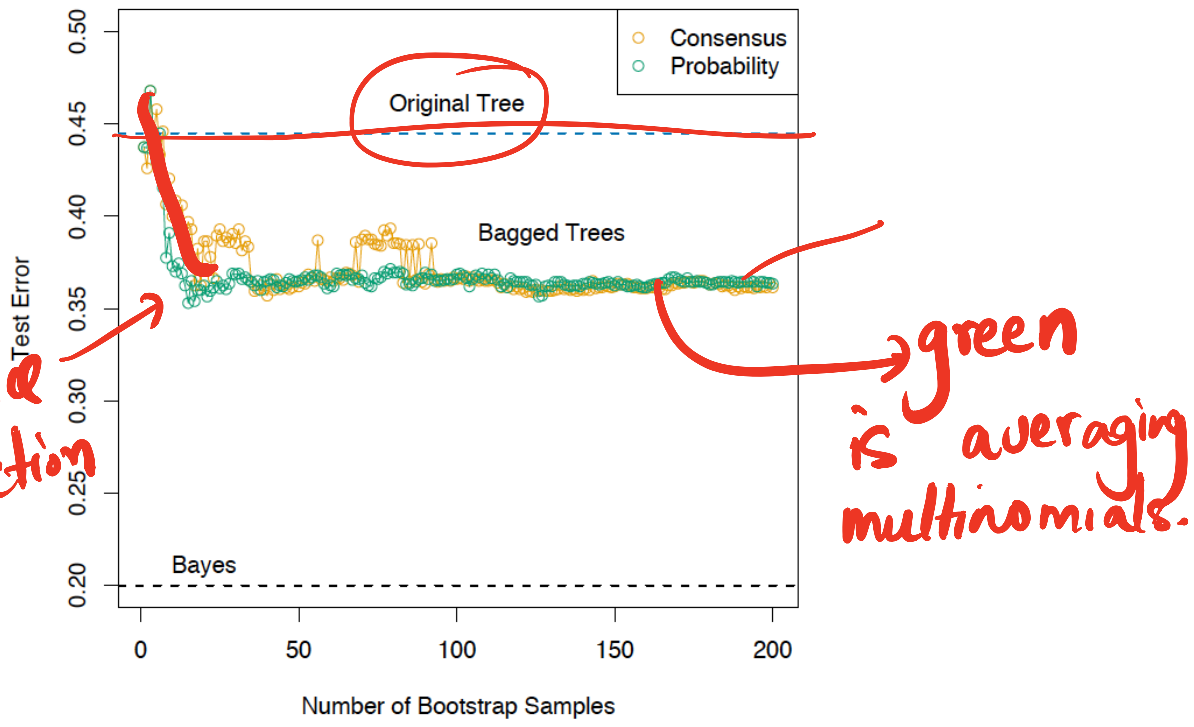
Example: bagging

Example (from ESL 8.7.1): $n = 30$ training data points, $p = 5$ features, and $K = 2$ classes. No pruning used in growing trees:



a fairly diverse set of trees.

Bagging helps decrease the misclassification rate of the classifier (evaluated on a large independent test set). Look at the orange curve:



Voting probabilities are not estimated class probabilities

Suppose that we wanted estimated class probabilities out of our bagging procedure.

What about using the proportion of votes that were for class k?

$$\boxed{\hat{p}_k^{\text{bag}}(x)} = \frac{1}{B}\sum_{b=1}^{B} 1\{\hat{f}^{\text{tree},b}(x) = k\}$$

what fraction said label k.

This is generally not a good estimate.

Simple example: suppose that the true probability of class 1 given x is 0.75. Suppose also that each of the bagged classifiers $\hat{f}^{\text{tree},b}(x)$ correctly predicts the class to be 1. Then $\hat{p}_1^{\text{bag}}(x) = 1$, which is wrong

Estimated class probabilities

each tree for a pt x $\left(\hat{\mathbb{P}}(Y=1|x), \hat{\mathbb{P}}(Y=2|x) \ldots\right.$

What's nice about trees is that each tree already gives us a set of predicted class probabilities at x:

$$\underline{\widehat{p}_k^{\,\text{tree},b}(x),}$$

These are simply the proportion of points in the appropriate region that are in each class

This suggests an alternative method for bagging. Now given an input $x \in \mathbb{R}^p$, instead of simply taking the prediction $\widehat{f}^{\,\text{tree},b}(x)$ from each tree, we go further and look at its predicted class probabilities $\widehat{p}_k^{\,\text{tree},b}(x)$, $k = 1, \ldots K$, and combine those!

Alternative form of bagging

We define the bagging estimates of class probabilities:

$$\underline{\widehat{p}_k^{\text{bag}}(x)} = \underline{\frac{1}{B}\sum_{b=1}^{B}\widehat{p}_k^{\text{tree},b}(x)} \quad k = 1,\dots K$$

→ don't average majority votes but average the probs. directly.

The final bagged classifier just chooses the class with the highest probability:

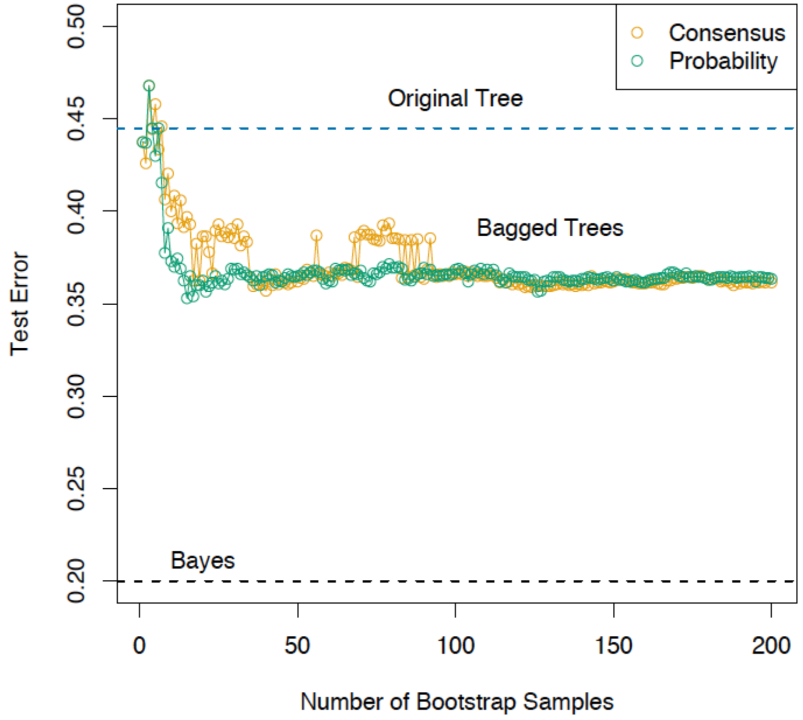
$$\underline{\widehat{f}^{\text{bag}}(x) = \underset{k=1,\dots K}{\text{argmax}}\ \widehat{p}_k^{\text{bag}}(x)}$$

This form of bagging is preferred if it is desired to get estimates of the class probabilities.

It also tends to give better predictions!

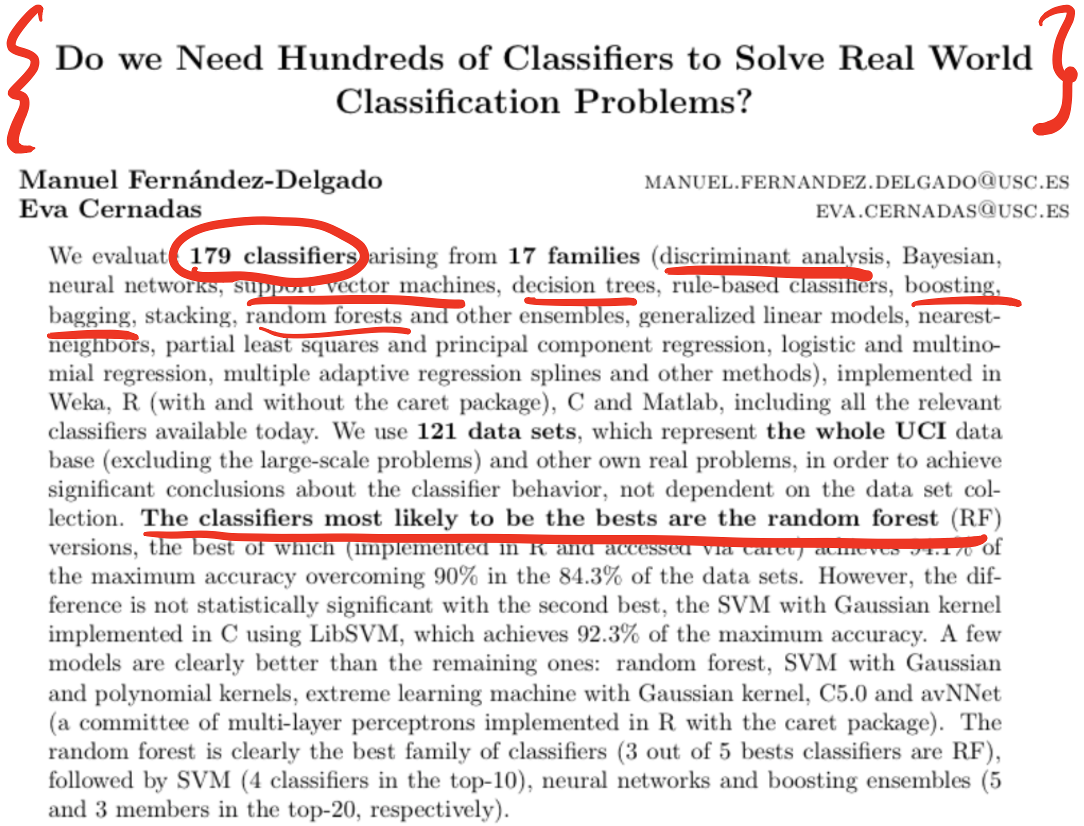
Example: alternative form of bagging

Previous example revisited: the alternative form of bagging produces misclassification errors shown in green



Random Forests: History and Motivation

- Invented by Leo Breiman while trying to understand boosting and bagging better.



Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?

Manuel Fernández-Delgado MANUEL.FERNANDEZ.DELGADO@USC.ES
Eva Cernadas EVA.CERNADAS@USC.ES

We evaluate **179 classifiers** arising from **17 families** (discriminant analysis, Bayesian, neural networks, support vector machines, decision trees, rule-based classifiers, boosting, bagging, stacking, random forests and other ensembles, generalized linear models, nearest-neighbors, partial least squares and principal component regression, logistic and multinomial regression, multiple adaptive regression splines and other methods), implemented in Weka, R (with and without the caret package), C and Matlab, including all the relevant classifiers available today. We use **121 data sets**, which represent **the whole UCI** data base (excluding the large-scale problems) and other own real problems, in order to achieve significant conclusions about the classifier behavior, not dependent on the data set collection. **The classifiers most likely to be the bests are the random forest** (RF) versions, the best of which (implemented in R and accessed via caret) achieves 94.1% of the maximum accuracy overcoming 90% in the 84.3% of the data sets. However, the difference is not statistically significant with the second best, the SVM with Gaussian kernel implemented in C using LibSVM, which achieves 92.3% of the maximum accuracy. A few models are clearly better than the remaining ones: random forest, SVM with Gaussian and polynomial kernels, extreme learning machine with Gaussian kernel, C5.0 and avNNet (a committee of multi-layer perceptrons implemented in R with the caret package). The random forest is clearly the best family of classifiers (3 out of 5 bests classifiers are RF), followed by SVM (4 classifiers in the top-10), neural networks and boosting ensembles (5 and 3 members in the top-20, respectively).

Random Forests: Improved Bagging

Random forests[3], an incredibly popular and successful prediction approach, are a $\underline{\text{simple modification of bagged trees}}$.

We've seen that bagging should improve with independence of the trees. Bagged trees can have dependence because the same strong variable always makes it into the same split.

Random forests make the trees more independent by only allowing a small subset of variables to be considered ***at each split***. At each split, $\boxed{m < p}$ variables are selected, and a $\underline{\text{split is chosen from}}$ $\underline{\text{among those variables}}$. $\sqrt{p}$.

more

This leads the trees to be more varied and ~~less~~ independent. It also leads to a nice balance among correlated variables. Finally, it actually gives better predictions!

[3] Breiman (2001), "Random Forests"

How do we estimate error?

How can we estimate how well we will predict/classify on new data?

Cross-validation is tempting, because we have no creativity.

However, cross-validation could be quite expensive, since we're boostrapping and developing hundreds of trees each time.

What about the samples each bootstrap did not select?

Out-of-Bag error estimation

Each training point should be missing from about $1/3$ of the bootstrap samples. That subset of the trees have not been overfit to the training point!

For each sample point, we get an average prediction from the $1/3$ of the trees that didn't include it. We combine these predictions over all the points to get an out-of-bag MSE estimate.

This acts like a cross-validated estimate, but it's free!

Digging into Random Forests

The bagged classifier is now an average of many trees. This is much harder to interpret! (If you thought trees were hard to interpret...)

Similarly, one of the big selling points of the Lasso is that it produces *interpretable* models. By seeing the sparsity pattern, we know which variables are being used for predicting.

We see that random forests predict incredibly well. However, an average of hundreds of random trees is harder to understand.

We need tools to peek into these “black box” methods to understand how they’re using the data.

Variable Importance

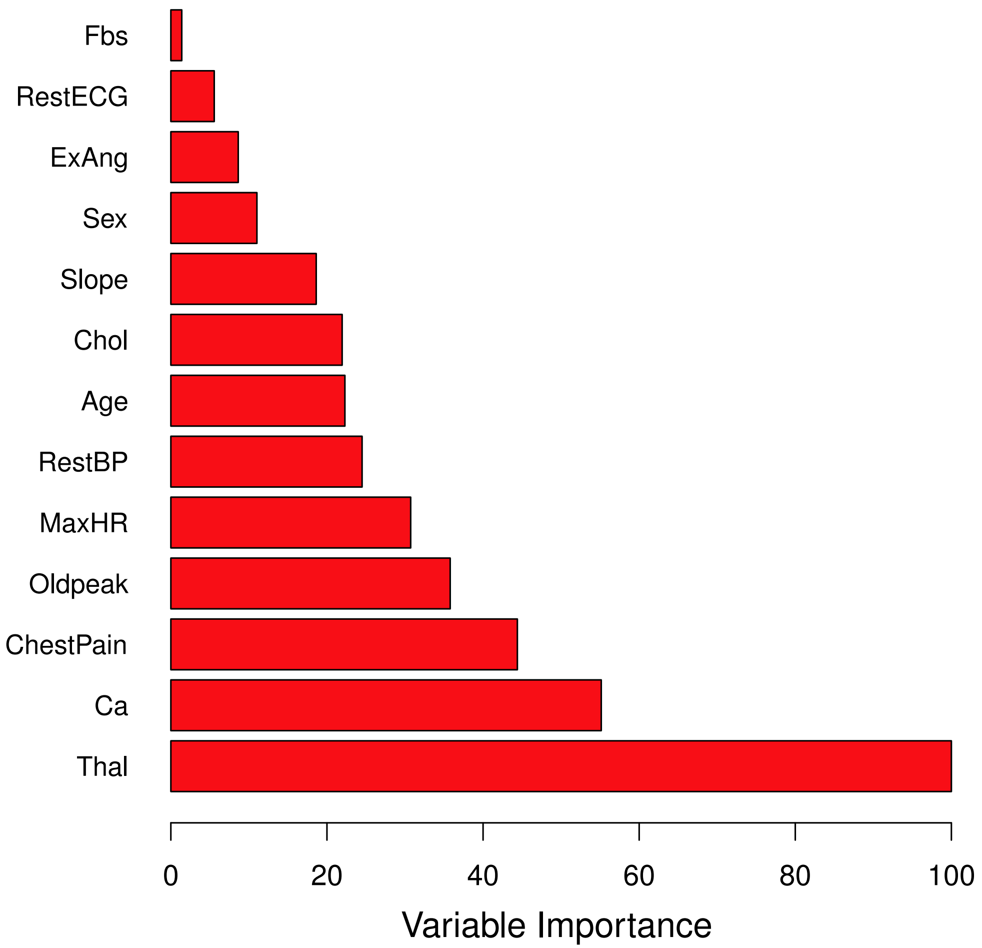
The most basic task is to understand which predictors are important for making the random forest prediction.

Random forests don't explicitly do variable selection like the lasso, but their individual trees tend to rely more on informative variables when they search for the next split.

There are two popular ways to measure variable importance:

1. For each variable, measure the amount that RSS (or Gini index) decreases due to splits in that variable. Average this over all trees in the forest
2. Randomly permute each variable (one at a time) and see how much the model performance decreases.

The `varImpPlot` method in `R` computes the first of these.


Fbs
RestECG
ExAng
Sex
Slope
Chol
Age
RestBP
MaxHR
Oldpeak
ChestPain
Ca
Thal
0
20
40
60
80
100
Variable Importance

Disadvantages

It is important to discuss some disadvantages of bagging:

- *Loss of interpretability:* the final bagged classifier is not a tree, and so we forfeit the clear interpretative ability of a classification tree
- *Computational complexity:* we are essentially multiplying the work of growing a single tree by B (especially if we are using the more involved implementation that prunes and validates on the original training data)