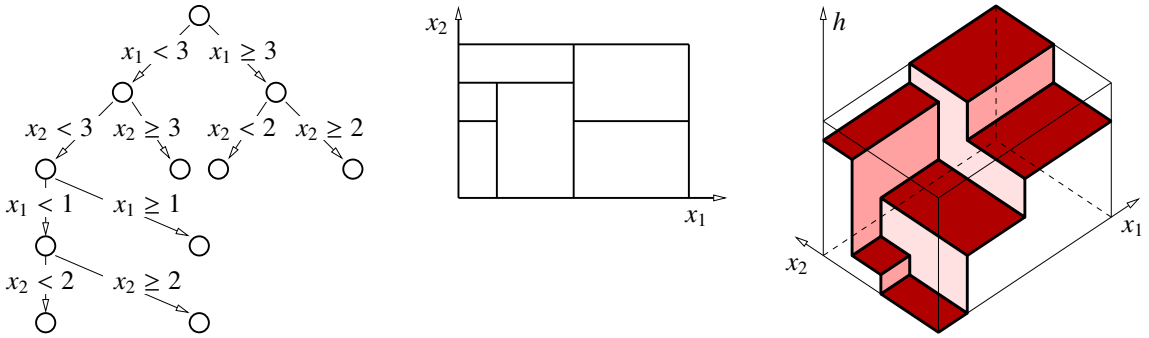
15 More Decision Trees, Ensemble Learning, and Random Forests

DECISION TREE VARIATIONS

[Last lecture, I taught you the vanilla algorithms for building decision trees and using them to classify test points. There are many variations on this basic algorithm; I'll discuss a few now.]

Decision Tree Regression

Creates a piecewise constant regression fn. [This seems too rudimentary to be true, but it's true.]



treeregresstree.pdf [Decision tree regression.]

Leaf stores label $\mu_S = \frac{1}{|S|}\sum_{i \in S} y_i$, the mean label for training pts $i \in S$.

Cost $J(S) = \text{Var}(\{y_i : i \in S\}) = \frac{1}{|S|}\sum_{i \in S}(y_i - \mu_S)^2$.

[So if all the points in a leaf have the same y-value, then the cost is zero.]
[We choose the split that minimizes the weighted average of the variances of the children after the split.]

Stopping Early

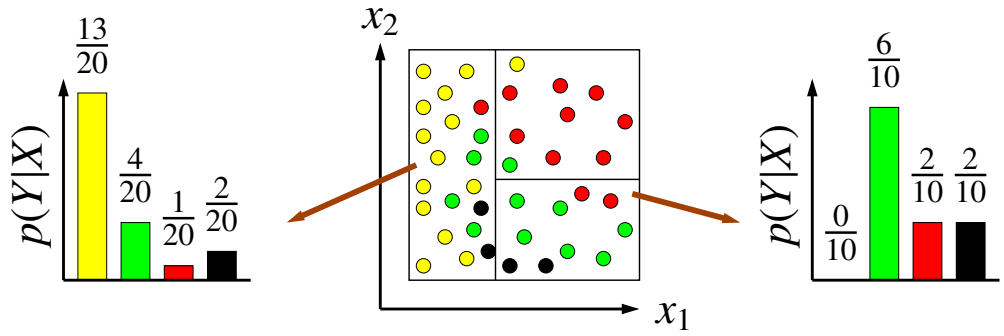
[The basic version of the decision tree algorithm keeps subdividing treenodes until every leaf is pure. We don't have to do that; sometimes we prefer to stop subdividing treenodes earlier.]

Why?

- Limit tree depth (for speed)
- Limit tree size (big data sets)
- Pure tree may overfit
- Given noise or overlapping distributions, pure leaves tend to overfit; better to stop early and estimate posterior probs

[When you have strongly overlapping class distributions, refining the tree down to one training point per leaf is absolutely guaranteed to overfit, giving you a classifier akin to the 1-nearest neighbor classifier. It's better to stop early, then classify each leaf node by taking a vote of its training points; this gives you a classifier akin to a k-nearest neighbor classifier.]

[Alternatively, you can use the points to estimate a posterior probability for each leaf, and return that. If there are many points in each leaf, the posterior probabilities might be reasonably accurate.]



leaf2.pdf [In the decision tree at left, each leaf has multiple classes. Instead of returning the majority class, each leaf could return a posterior probability histogram, as illustrated at right.]

Leaves with multiple points return

- a majority vote or class posterior probs (classification) or
- an average (regression).

How to stop? Select stopping condition(s):

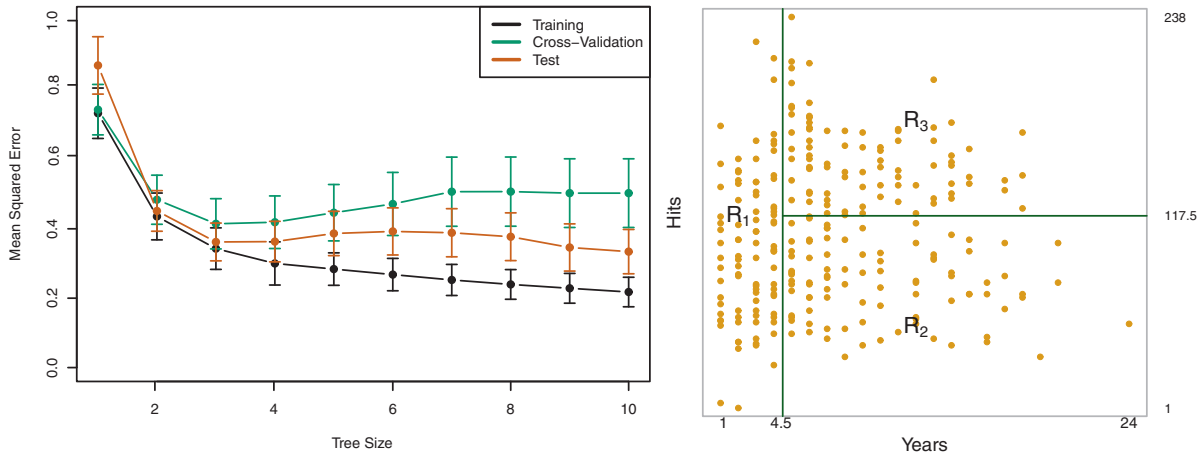
- Next split doesn't reduce entropy/error enough (dangerous; pruning is better)
- Most of node's points (e.g., > 90%) have same class [to deal with outliers & overlapping distribs]
- Node contains few training points (e.g., < 10) [especially for big data]
- Box's edges are all tiny [super-fine resolution may be pointless]
- Depth too great [risky if there are still many training points in the box]
- Use validation to compare

[The last is the slowest but most effective way to know when to stop: use validation to decide whether splitting the node lowers your validation error. But if your goal is to avoid overfitting, it's generally even more effective to grow the tree a little too large and then use validation to prune it back . . .]

Pruning

Grow tree too large; greedily remove each split whose removal improves validation performance.
[We have to do validation once for each split that we're considering reversing.]
More reliable than stopping early.

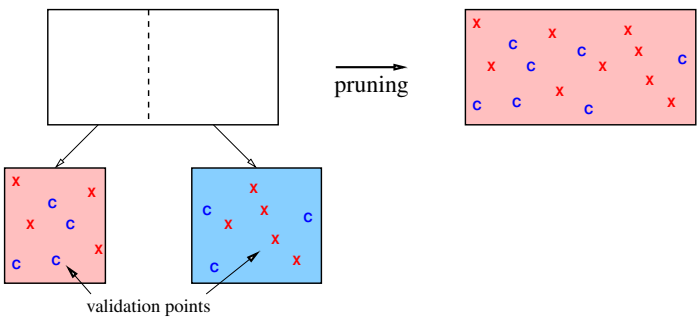
[The reason why pruning often works better than stopping early is because often a split that doesn't seem to make much progress is followed by a split that makes a lot of progress. If you stop early, you'll never find out. Pruning is simple and highly recommended when you have enough time to build and prune the tree.]



prunehitters.pdf, prunedhitters.pdf (ISL, Figures 8.5 & 8.2) [At left, a plot of decision tree leaf nodes vs. errors for baseball hitter data. At right, the best decision tree has three leaves. Players' salaries: R1 = \$165,174, R2 = \$402,834, R3 = \$845,346.]

[In this example, a 10-leaf decision tree was constructed to predict the salaries of baseball players, based on their years in the league and average hits per season. Then the tree was pruned by validation. The best decision tree on the validation data turned out to have just three leaves.]
[Observe that this tree is very *interpretable*. You could easily explain it to grandpa.]

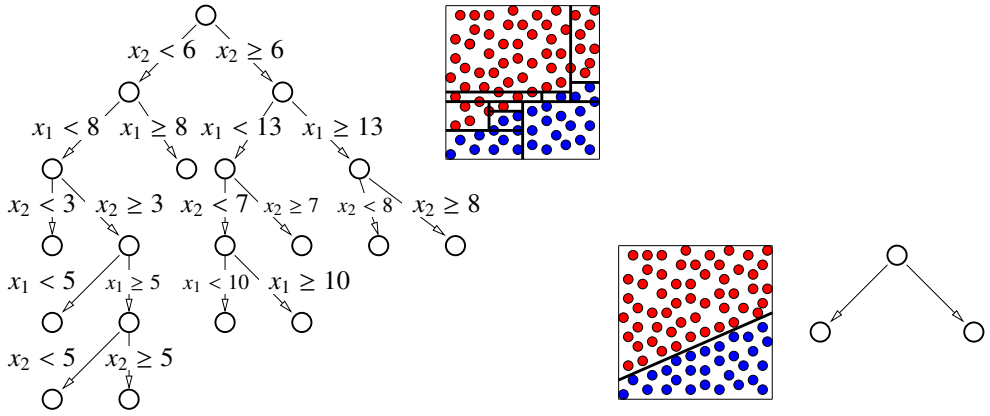
[It might seem expensive to do validation once for each split we consider reversing. But you can do it pretty cheaply. What you *don't* do is reclassify every training point from scratch. Instead, you first compute which leaf each validation point winds up in, then for each leaf you make a list of its validation points. When you are deciding whether to remove a split, you just look at the validation points in the two leaves you're thinking of removing, and see how they will be reclassified and how that will change the error rate. You can compute this very quickly.]



prunecheck.pdf [After we determine which leaf boxes each validation point ends up in, we find that pruning these two leaves improves the validation accuracy. Box colors indicate the majority classes of the training points (not shown). Local validation accuracy improves from 7/16 to 9/16.]

Multivariate Splits

Find non-axis-aligned splits with other classification algs or by generating them randomly.



multivar.pdf [An example where an ordinary decision tree needs many splits to approximate a diagonal linear decision boundary, but a single multivariate split takes care of it.]

[Here you can use other classification algorithms such as SVMs, logistic regression, and Gaussian discriminant analysis. Decision trees permit these algorithms to find nonlinear decision boundaries by making them hierarchical.]

May gain better classifier at cost of worse interpretability or speed.
[Standard decision trees are very fast because they check only one feature at each treenode. But if there are hundreds of features, and you have to check all of them at every level of the tree to classify a point, it slows down classification a lot.]
[A good compromise is to set a limit on the number of features you check at each treenode—say, three. You can use forward stepwise selection at each treenode to choose the three features.]
[On exams, assume we check only one feature per treenode unless we say otherwise!]

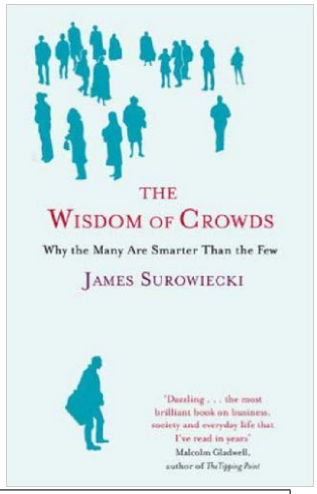
ENSEMBLE LEARNING

Decision trees are fast, simple, interpretable, easy to explain,
invariant under scaling/translation, robust to irrelevant features.

But not the best at prediction. [Compared to previous methods we've seen.]
High variance. [Though we can achieve very low bias.]

[For example, suppose we take a training data set, split it into two halves, and train two decision trees, one on each half of the data. It's not uncommon for the two trees to turn out very different. In particular, if the two trees pick different features for the very first split at the root of the tree, then it's quite common for the trees to be completely different. So decision trees tend to have high variance.]

[So let's think about how to fix this. As an analogy, imagine that you are generating random numbers from some distribution. If you generate just one number, it might have high variance. But if you generate n random numbers and take their average, then the variance of that average is n times smaller. So you might ask yourself, can we reduce the variance of decision trees by taking an average answer of a bunch of decision trees? Yes we can.]



wisdom.jpg, penelope.jpg [James Surowiecki's book "The Wisdom of Crowds" and Penelope the cow. Surowiecki tells us this story . . .]

[A 1906 county fair in Plymouth, England had a contest to guess the weight of an ox. A scientist named Francis Galton was there, and he did an experiment. He calculated the median of everyone's guesses. The median guess was 1,207 pounds, and the true weight was 1,198 pounds, so the error was less than 1%. Even the cattle experts present didn't estimate it that accurately.]
[National Public Radio repeated the experiment in 2015 with a cow named Penelope whose photo they published online. They got 17,000 guesses, and the average guess was 1,287 pounds. Penelope's actual weight was 1,355 pounds, so the crowd got it to within 5 percent.]
[The main idea is that sometimes the average opinion of a bunch of idiots is better than the opinion of one expert. And so it is with learning algorithms. We call a learning algorithm a weak learner if it does better than guessing randomly. And we combine a bunch of weak learners to get a strong one.]

We can take average of output of
- different learning algs
- same learning alg on many training sets [if we have tons of data]
- bagging: same learning alg on many random subsamples of one training set
- random forests: randomized decision trees on random subsamples

[These last two are the most common ways to use averaging, because usually we don't have enough training data to use fresh data for every learner.]
Metalearner takes test point, feeds it into all T learners, returns majority class or average output.

[Averaging is not specific to decision trees; it can work with many different learning algorithms. But it works particularly well with decision trees.]

Regression algs: take median or mean output [of all the weak learners]
Classification algs: take majority vote OR average posterior probs

[Apology to readers: I show some videos in this lecture, which cannot be included in this report.]

[Show averageaxis.mov] [Here's a simple classifier that takes an average of "stumps," trees of depth 1. Observe how good the posterior probabilities look.]
[Show averageaxistree.mov] [Here's a 4-class classifier with depth-2 trees.]

[The Netflix Prize was an open competition for the best collaborative filtering algorithm to predict user ratings for films, based on previous ratings. It ran for three years and ended in 2009. The winners used an extreme ensemble method that took an average of many different learning algorithms. In fact, a couple of top teams combined into one team so they could combine their methods. They said, "Let's average our models and split the money," and that's what happened.]

Use learners with low bias (e.g., deep decision trees).
High variance & some overfitting are okay. Averaging reduces the variance!
[Each learner may overfit, but each overfits in its own unique way.]
Averaging sometimes reduces bias & increases flexibility a bit, but not reliably.
e.g., creating nonlinear decision boundary from linear classifiers.
[Averaging rarely reduces bias as much as it reduces variance, so get the bias small before averaging.]

Hyperparameter settings usually different than 1 learner. [Averaging reduces variance more than bias.]

[Sometimes the number of trees is said to be a hyperparameter, but it's not really true. When random forests work, usually the variance drops monotonically with the number of trees, and the main limit on the number of trees is computation time. The stopping conditions are the hyperparameters.]

Bagging = Bootstrap AGGregatING (Leo Breiman, 1994)

[Leo Breiman was a statistics professor right here at Berkeley. He did his best work after he retired in 1993. The bagging algorithm was published the following year, and then he went on to co-invent random forests as well. Unfortunately, he died in 2005.]

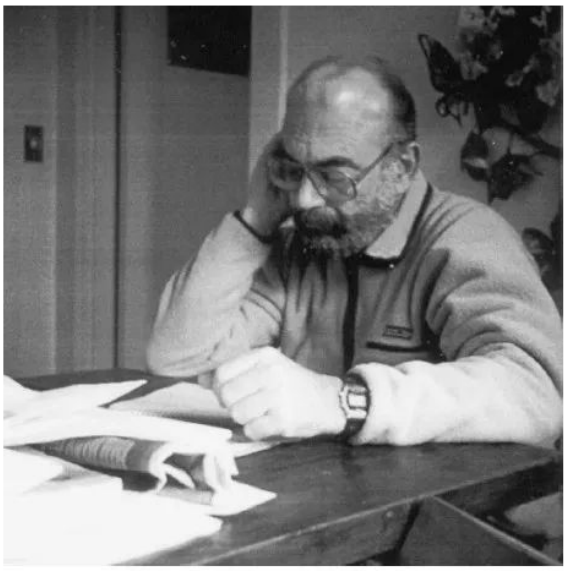
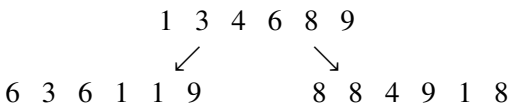


FIG. 6. *Leo working in a prior Berkeley residence*, 1985.

leobreiman3.png [Leo Breiman]

[Bagging is a randomized method for creating many different learners from the same data set. It works well with many different learning algorithms. One exception seems to be k-nearest neighbors; bagging mildly degrades it.]
Given n-point training sample, generate random subsample of size n' by sampling *with replacement*. Some points chosen multiple times; some not chosen.



If $n' = n$, $\sim 63.2\%$ are chosen. [On average; this fraction varies randomly.]
Build learner. Points chosen j times have greater weight:
[If a point is chosen j times, we want to treat it the same way we would treat j different points all bunched up infinitesimally close together.]

- Decision trees: j-time point has $j \times$ weight in entropy.
- SVMs: j-time point incurs $j \times$ penalty to violate margin.
- Regression: j-time point incurs $j \times$ loss.

Repeat until T learners.

Random Forests

Random sampling isn't random enough!
[With bagging, often the decision trees look very similar. Why is that?]
One really strong predictor → same feature split at top of every tree.
[For example, if you're building decision trees to identify spam, the first split might always be "viagra." Random sampling might not change that. If the trees are very similar, then taking their average doesn't reduce the variance much.]
Let's reduce the correlation between different trees.

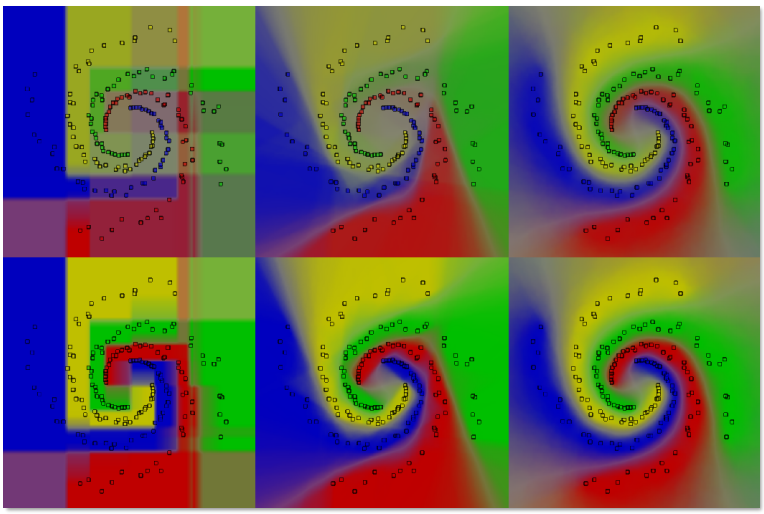
Idea: At each treenode, take random sample of m features (out of d).
Choose best split from m features.
[We're not allowed to split on the other $d - m$ features!]
Different random sample for each treenode.
$m \approx \sqrt{d}$ works well for classification; $m \approx d/3$ for regression.
[So if you have a 100-dimensional feature space, you randomly choose 10 features and pick the one of those 10 that gives the best split. But m is a hyperparameter, and you might get better results by tuning it for your particular application. These values of m are good starting guesses.]
Smaller $m \rightarrow$ more randomness, less tree correlation, more bias

[One reason this works is if there's a really strong predictor, only a fraction of the trees can choose that predictor as the first split. That fraction is m/d. So the split tends to "decorrelate" the trees. And that means that when you take the average of the trees, your average will have less variance than a single tree.]
[You have to be careful, though, because you don't want to dumb down the trees too much in your quest for decorrelation. Averaging works best when you have very strong learners that are also diverse. But it's hard to create a lot of learners that are very different yet all very smart. The Netflix Prize winners did it, but it was a huge amount of work.]

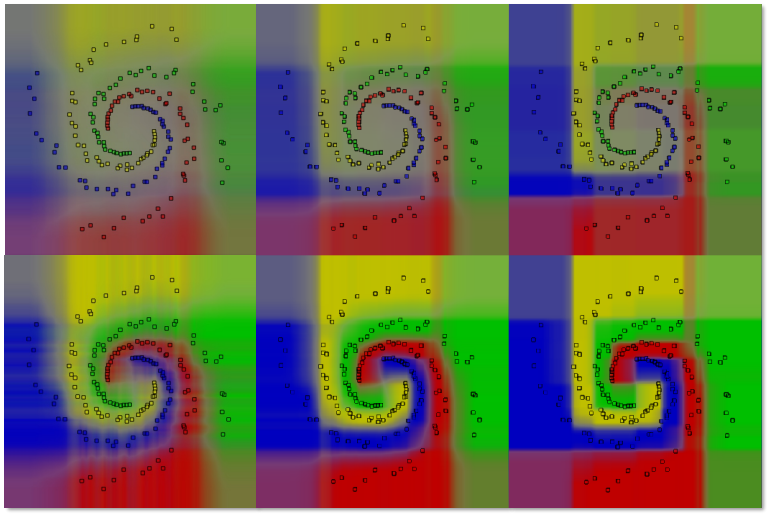
Sometimes test error drops even at 100s or 1,000s of decision trees!
Disadvantages: slow; loses interpretability/inference.
[But the compensation is it's more accurate than a single decision tree.]

[I will end by showing you examples of a very non-standard method for random forests that works magic in certain difficult circumstances.]
Idea: generate s random multivariate splits (oblique lines, quadrics); choose best split.
[You have to be a bit clever about how you generate random decision boundaries; I'm not going to discuss that. I'll just show lots of examples.]

[Show treesidesdeep.mov] [Lots of good-enough conic random decision trees.]
[Show averageline.mov]
[Show averageconic.mov]
[Show square.mov] [Depth 2; look how good the posterior probabilities look.]
[Show squaresmall.mov] [Depth 2; see the uncertainty away from the center.]
[Show spiral2.mov] [Doesn't look like a decision tree at all, does it?]
[Show overlapdepth14.mov] [Overlapping classes. This example overfits!]
[Show overlapdepth5.mov] [Better fit.]



500.pdf [Random forest classifiers for 4-class spiral data. Each forest takes the average of 400 trees. The top row uses trees of depth 4. The bottom row uses trees of depth 12. From left to right, we have axis-aligned splits, splits with lines with arbitrary rotations, and splits with conic sections. Each split is chosen to be the best of 500 random choices.]



randomness.pdf [Random forest classifiers for the same data. Each forest takes the average of 400 trees. In these examples, *all* the splits are axis-aligned. The top row uses trees of depth 4. The bottom row uses trees of depth 12. From left to right, we choose each split from 1, 5, or 50 random choices. The more choices, the less bias and the better the classifier.]