Random Forests™

Ensembles of bootstrapped, weakened trees

Terence Parr
MSDS program
University of San Francisco

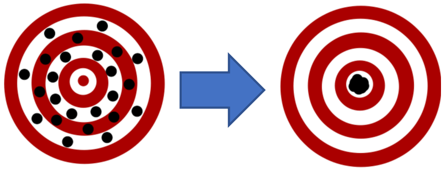
UNIVERSITY OF SAN FRANCISCO

Leo Breiman (1996) introduced bagging
then Random Forests (2001)
https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf

RF motivation

- Decision trees can often get training errors close to zero because we can grow very large trees to partition the feature space into tiny regions with 1 or just a few observations/samples; trees are very accurate on the training set and have very **low bias**
- The downside is that decision trees overfit like mad; or, using stats nerd terminology, decision trees have **high variance** and don't generalize well
- High variance implies model parameters vary a lot if we tweak the training data just a little bit

How can we increases generality?



- **Goal**: keep the high accuracy, but increase the generality
- Recall: simplifying models can increase generality at cost of some error
- So, let's weaken our decision tree model but in a way that makes predictions **noisy not biased**
- That means the model's prediction might be too high for one test case and too low in another
  - But, the model will not always be too low or too high, which would be biased
  - The expected value of a weakened model's prediction is same as that of full strength model
- To compensate for the noise and claw back some accuracy, make an ensemble of such weakened trees: the ensemble is **accurate on average**
- Ensemble predictions are the aggregate of the trees' predictions (average prediction or majority vote)

Target image credit: http://ficscience.blogspot.com/2011/02/technicalities-of-technical-terms.html

The key trick is amnesia

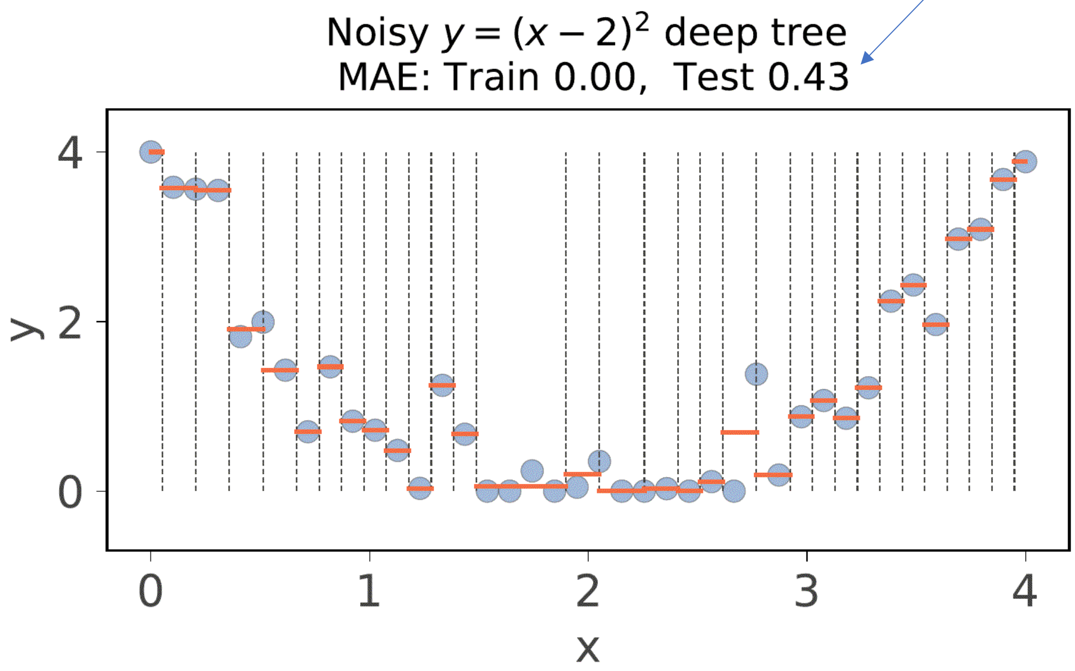
- Random forests are all about adding a bit of amnesia
- We will weaken trees by training each on a randomly selected subset of the training data: **bagged trees**
- Further, we will have training purposely forget about some features as we create decision nodes: **random forests**

Analogy: Crowdsourcing SF house prices

- Recruit multiple real estate agents to build house price models in their heads by visiting lots of houses; then each agent can estimate prices of unvisited houses
- Agents choose and examine house subsets **independently**
- There will be some overlap in visited houses sets but the subsets will be *independent and identically distributed* (*i.i.d.*)
- An agent trained on all houses is both accurate and general
- An agent trained on an i.i.d. subset is not biased *per se* but less accurate—a prediction for one house might be too low but a prediction for another house might be too high (avoiding bias)
- The variance of the ensemble average will be much tighter than the variance of an individual tree's prediction
- Averaging all agents' predictions reduces noise and is unbiased

Ex: Overfit decision trees regressors

- Animation shows 1D feature space partitioning of i.i.d. sample sets
- Slightly different training data sets can yield very different decision trees
- Clearly the trees have gotten way too specific to the data set
- Notice how the training error is 0 but (20% hold out) test error is terrible!



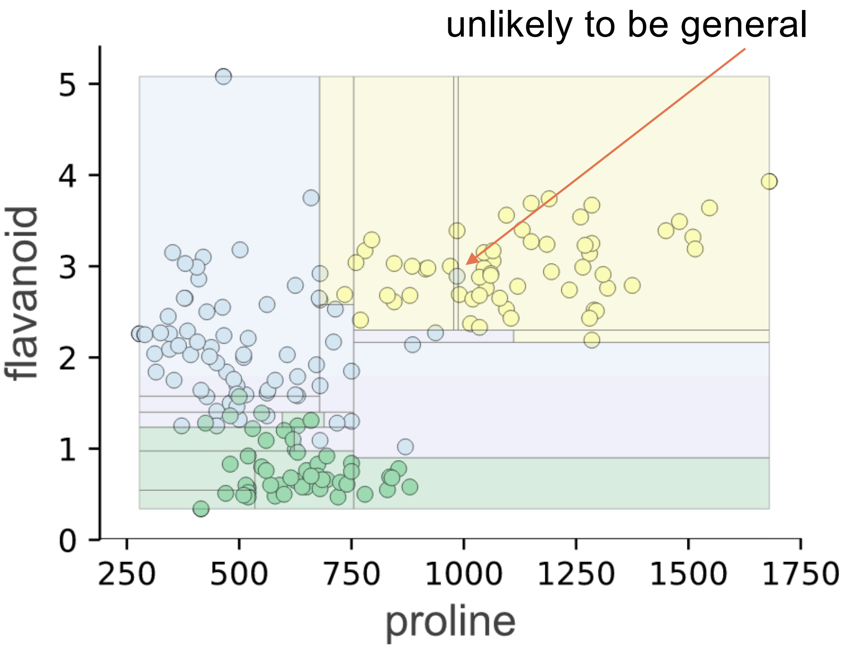
Uh oh!

See https://github.com/parrt/msds621/blob/master/notebooks/trees/random-forests.ipynb

UNIVERSITY OF SAN FRANCISCO

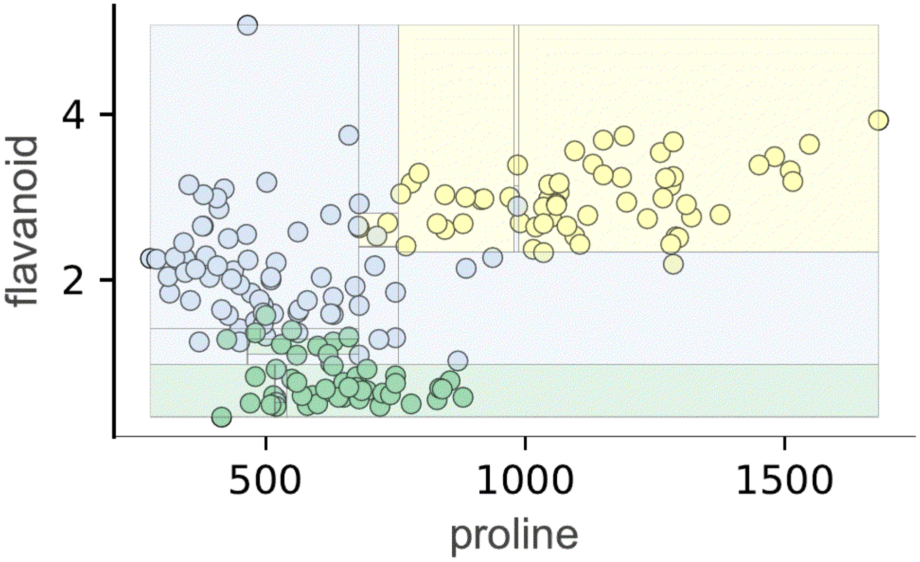
Ex: Overfit decision tree classifiers

- Here is a previous example where partitioning trapped a lonely blue in a sea of yellow
- In practice, we're given just one data set so let's do some sampling to get some i.i.d. "copies"
- Then see how different data sets give different partitioning (from different trees)



Partitioning from bootstrapped data

- *Bootstrap* (X, y) to simulate multiple i.i.d. data sets; each set gets ~63% of unique (X, y) data (sample n records with replacement)
- Animation shows 2D feature space partitions from various bootstraps
- Partitioning clearly varies a lot between bootstraps
- *OOB* == "out of bag" (more later)



Bagged tree classifier partitioning
Accuracy: train 100.0% OOB 85.7%
Showing all (X, y) data points

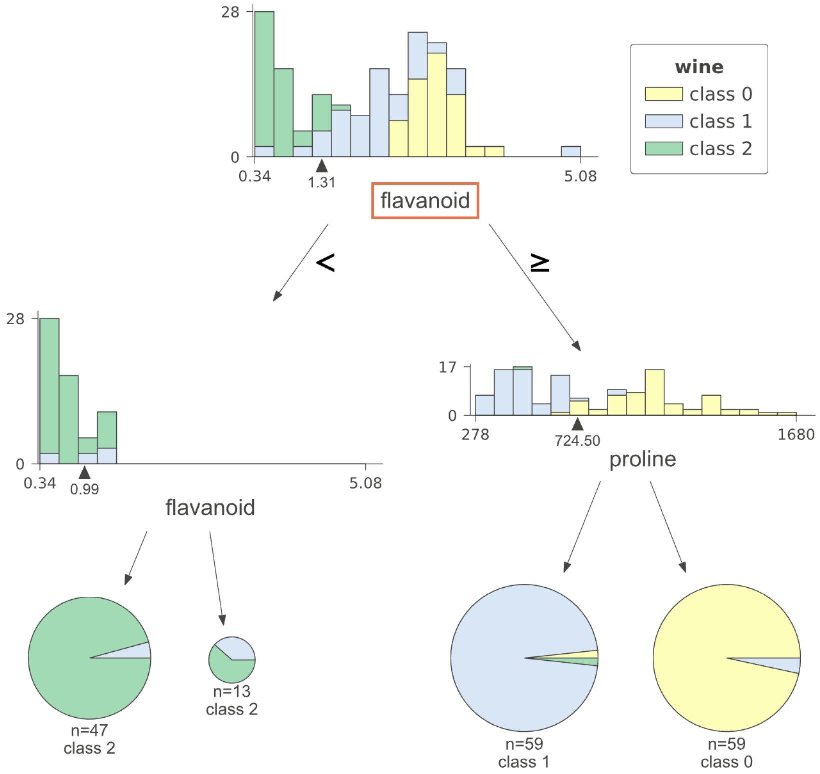
Uh oh!
Varies

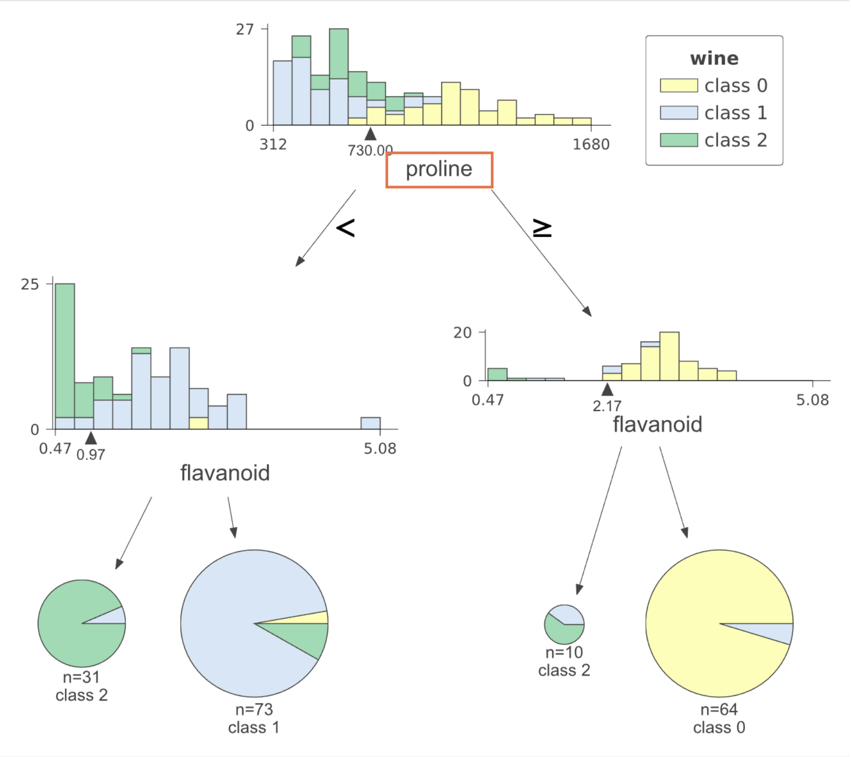
See https://github.com/parrt/msds621/blob/master/notebooks/trees/random-forests.ipynb

UNIVERSITY OF SAN FRANCISCO

Bootstrapping gives slightly different trees

`DecisionTreeClassifier(max_depth=2)`





Aside: Code for bootstrapping

NumPy

```
# Bootstrap: sample with replacement
n = len(y)
idx = np.random.randint(0,n,size=n)
X_train = X[idx]
y_train = y[idx]
```

```
# get OOB (out-of-bag) samples
mask = np.ones(n, dtype=bool)
mask[idx] = False
X_test = X[mask]
y_test = y[mask]
```

Pandas

```
# If data in dataframe
df = df.sample(len(df), replace=True)
```

See https://github.com/parrt/msds621/blob/master/notebooks/trees/random-forests.ipynb

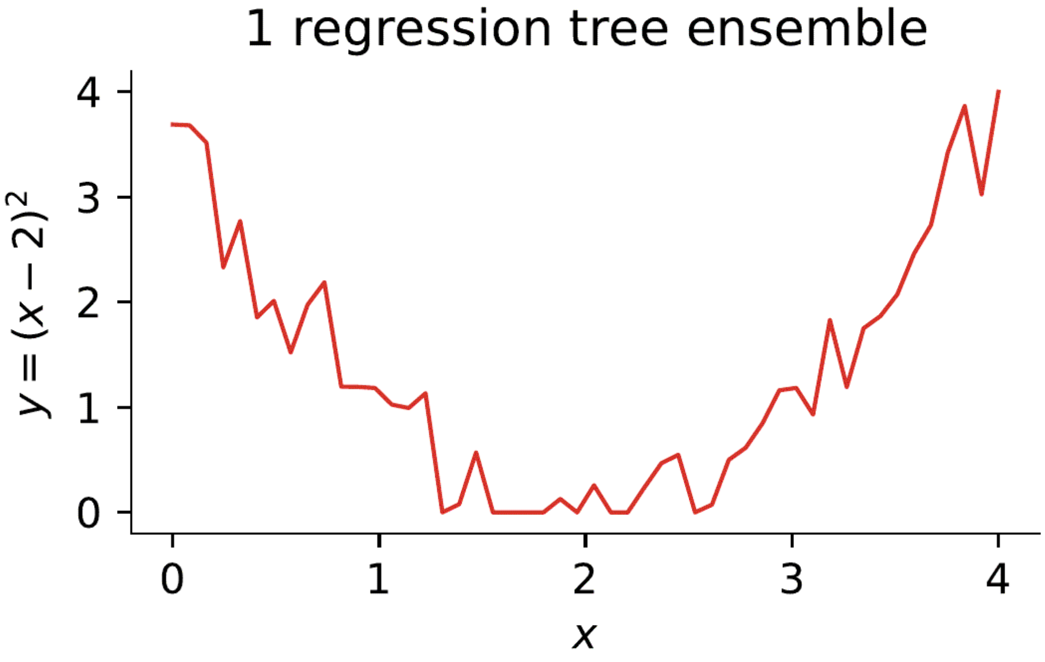
Bagged trees

Training trees on bootstrapped samples and aggregating predictions

UNIVERSITY OF SAN FRANCISCO

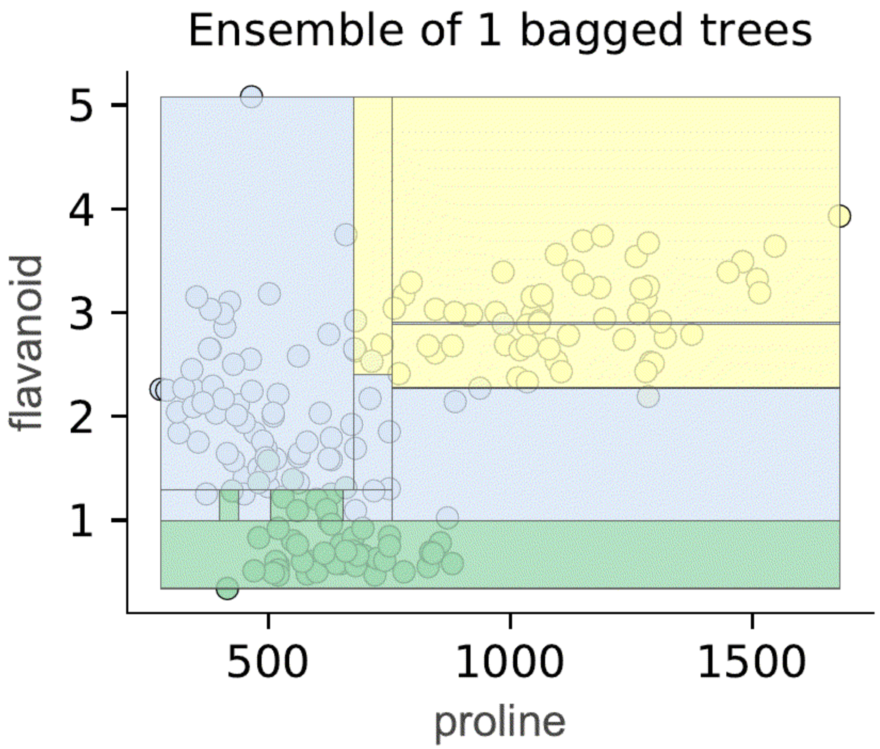
Ensemble of high-variance regression trees

- Animation shows how averaging the prediction of an ensemble of overfit trees actually produces a reasonable combined prediction
- As we add trees, the average prediction (red line) smooths out to the underlying quadratic distribution from which we draw noisy samples
- Note: variance of individual tree predictions stays high as we add trees, but the variance of the ensemble average tightens



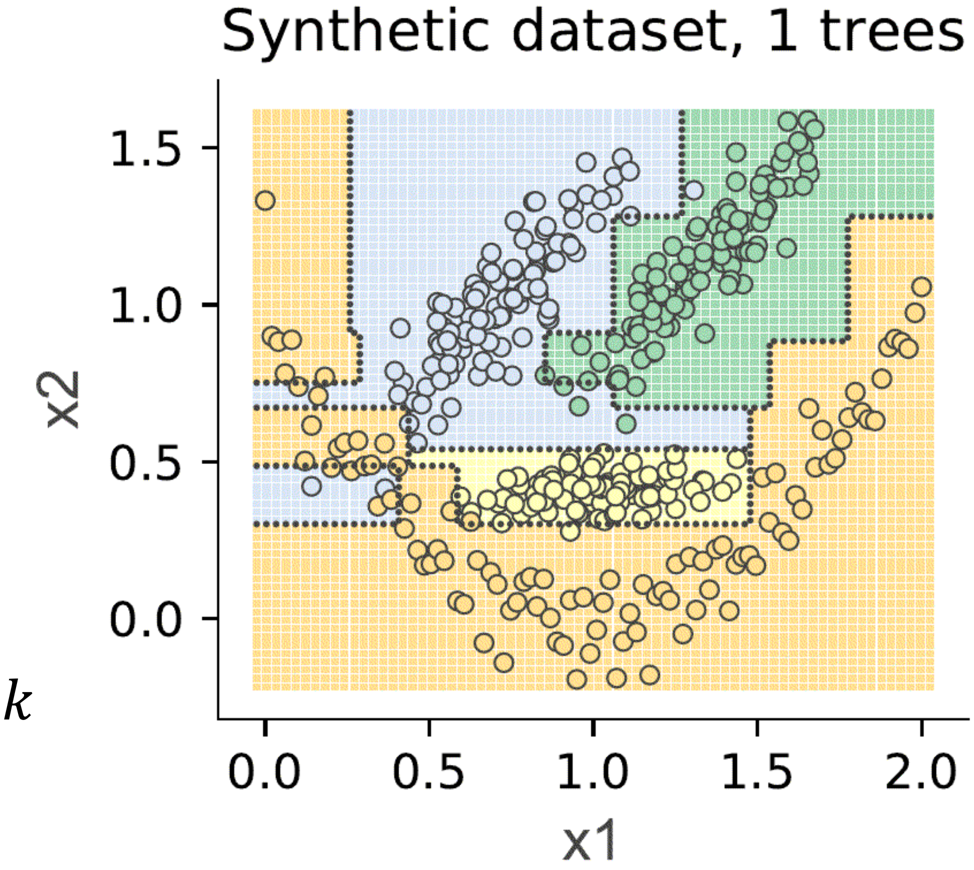
Ensemble of high-var. classification trees

- Animation shows overlapping prediction regions from multiple classifier trees
- Training data for each tree is bootstrapped from the original (X, y) data
- As we add trees, the averaged prediction regions become more stable and the decision boundaries more complex
- "Bag" is *bootstrap aggregation*



Ensemble classifier on synthetic data set

- Animation shows prediction regions from multiple bagged classifier trees
- Colored tiles indicate the probabilities of the various classes; e.g., yellow-orange color indicates uncertainty between those two classes
- Probability of class k at tile is proportion of trees that predict k



Animation uses experimental function I'm adding to dtreeviz

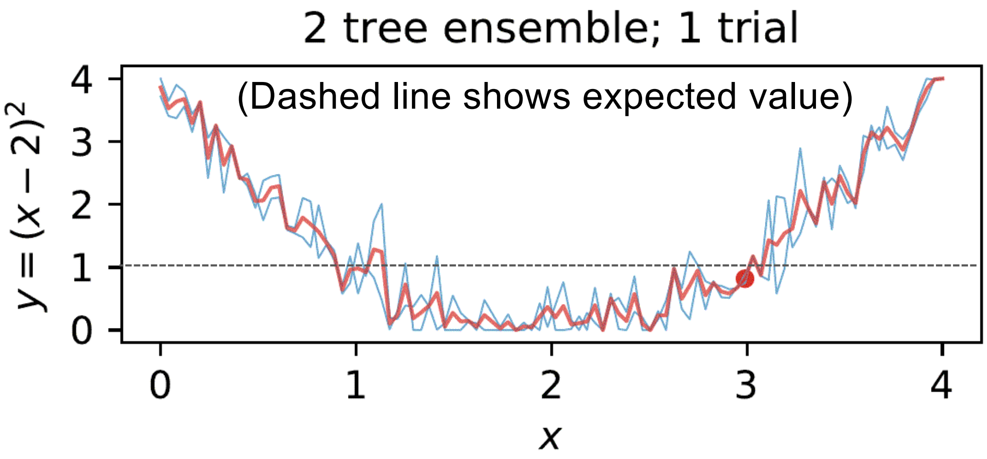
Ensemble's effect on bias and variance

- Train T trees on T i.i.d. X data sets
- Central limit theorem says that if variance of an i.i.d. random variable is σ^2, the variance of the average of T such vars is σ^2/T
- So, as we add trees, the variance of the ensemble prediction will shrink, which means we squeeze out noise
- The average of the tree predictions is the same as the <u>expected prediction</u> from any tree trained on one of the X sets (since i.d.)
- Each tree gives noisy predictions, sometimes too low and sometimes too high, but is unbiased

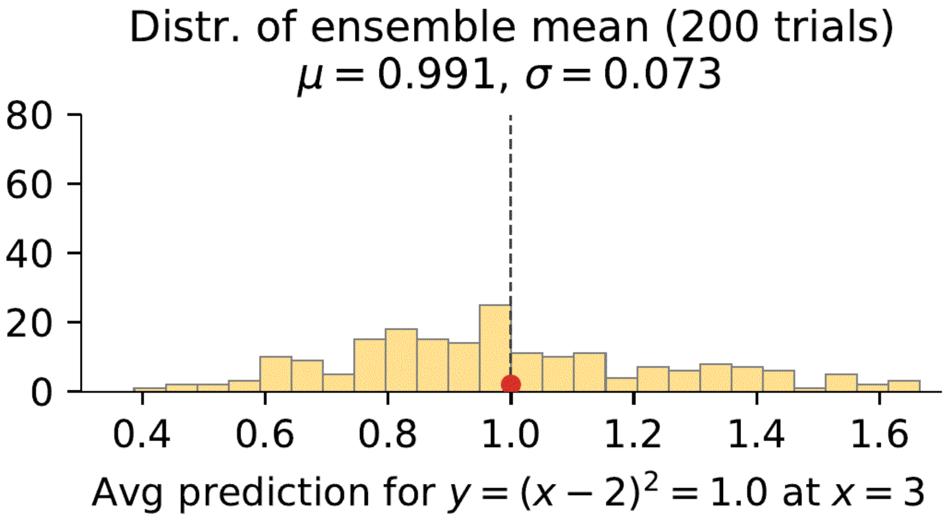
See page 588 ESLII book

Ex: variance of ensemble prediction

- Animation shows tree and ensemble predictions on left for T trees; magnitude of noise in blue tree predictions doesn't change with the number of trees but red line get less noisy with more trees
- At $x = 3$, expected value of ensemble is 1.0; create 200 separate ensembles of size T and compute variance of ensemble predictions at $x = 3$; distribution of ensemble average shown on the right


2 tree ensemble; 1 trial

(Dashed line shows expected value)


Distr. of ensemble mean (200 trials)
$\mu = 0.991$, $\sigma = 0.073$

Problem: trees are not independent thinkers

- With real estate agent analogy, we implicitly assumed agents were independent thinkers, and not clones
- But, decision trees are like robot clones and, given the same bit of data, yield the same bit of tree
- Imagine worst case: bootstrapping yields T identical sets so ensemble gives exactly the same prediction as any single tree
- In practice, if there is one strongly predictive var out of p, then all trees would be similar; initial root splits, and many others, would likely be same

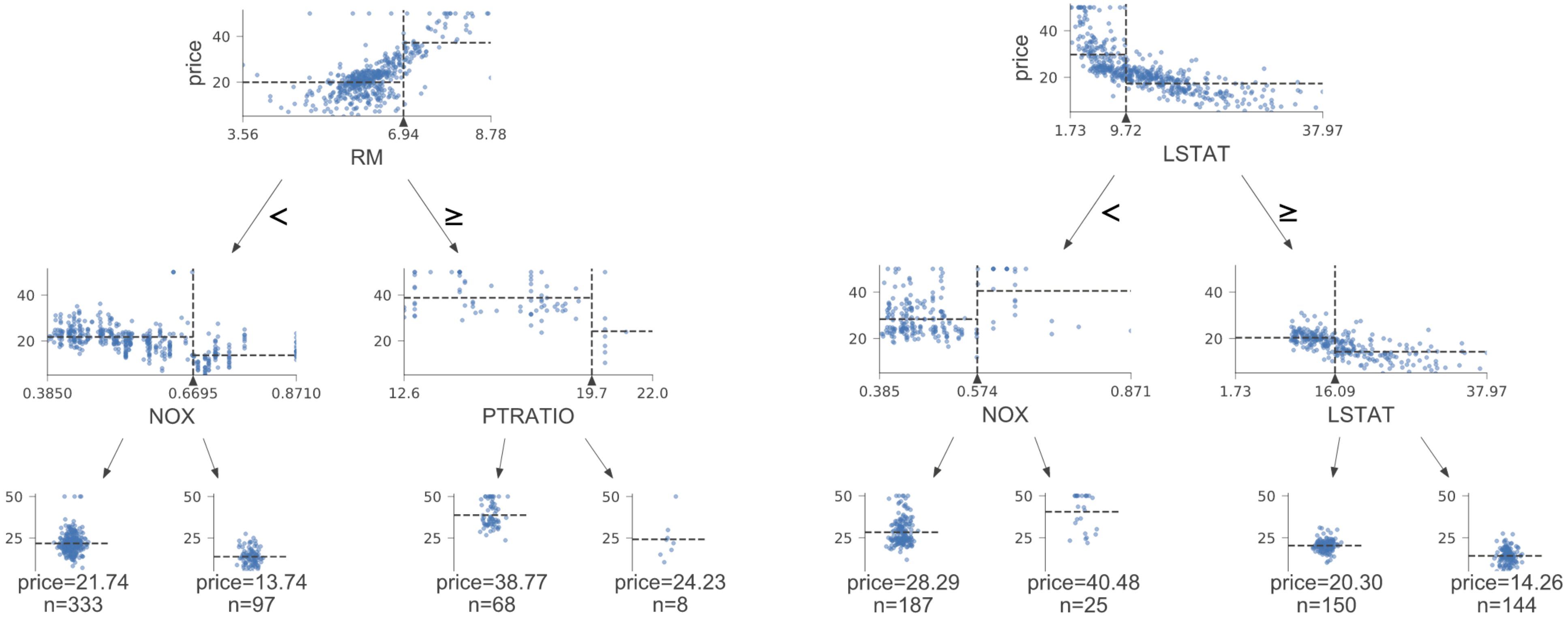
Random Forests

Ensembles of de-correlated bagged trees

Making trees independent thinkers

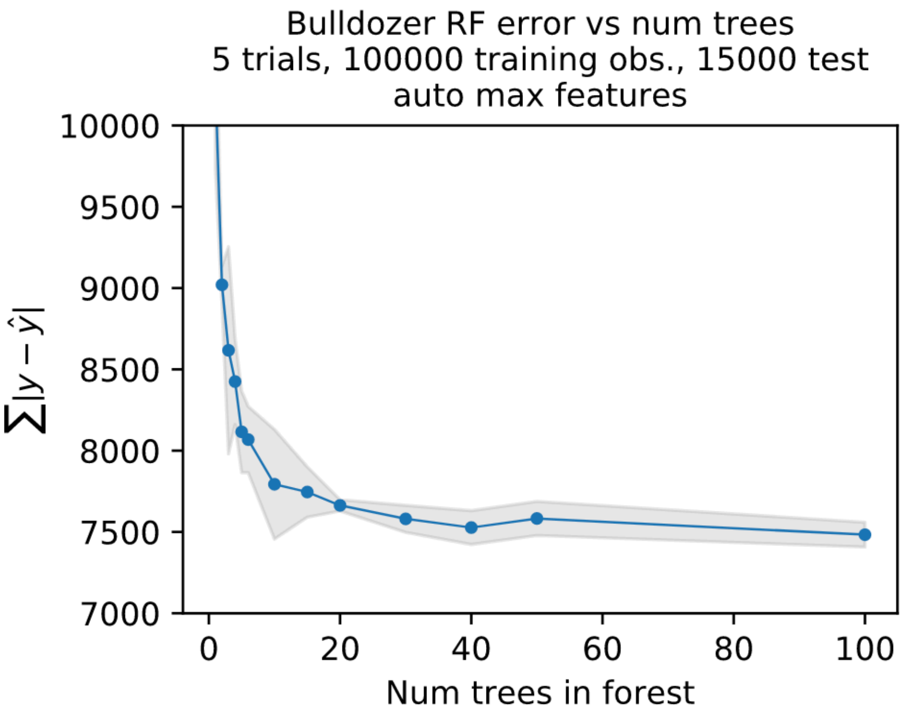
- Bagging overcomes most of the overfitting but we can improve generality a little by further weakening the trees in an effort to make them think more independently
- Restrict the available features when searching for a decision node split; choose from m randomly selected features
- Choose max features per split, $m \leq p$, such as sqrt(p)
- Make sure chance of selecting predictive variables (m/p) is high enough to find predictor variables (See ESLII p596)
- Let validation error be your guide to choosing m
- A random forest is then just: an ensemble of decision trees trained on bootstraps and whose feature selection strategy has a bit of amnesia

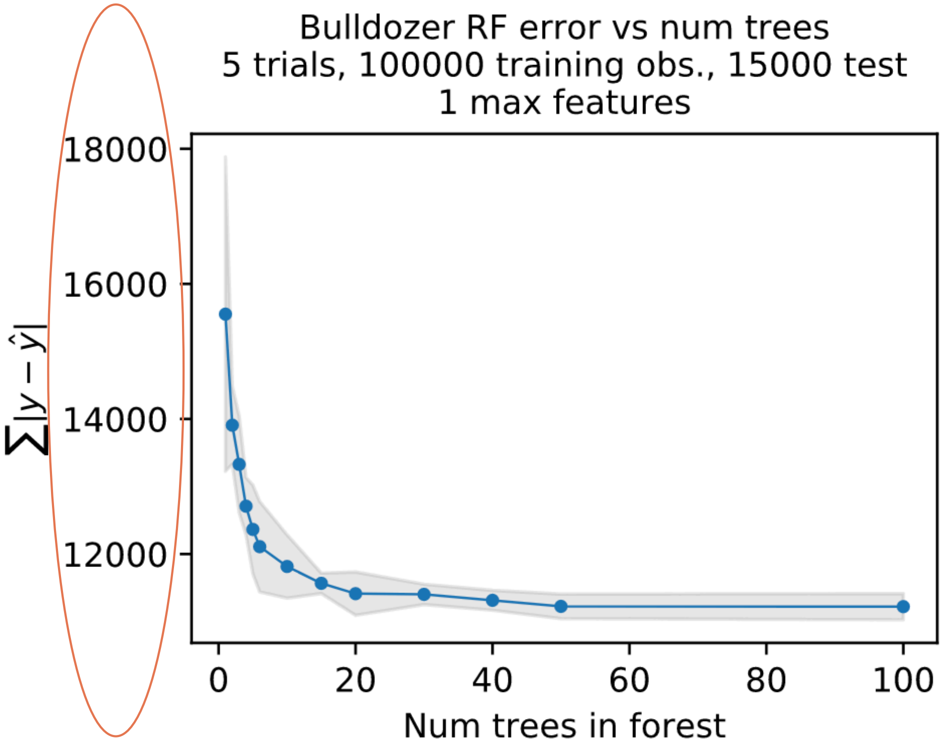
Ex: 2 trees trained on entire Boston set with m=5 (of 13)



Choose from 5 randomly selected features during EACH split

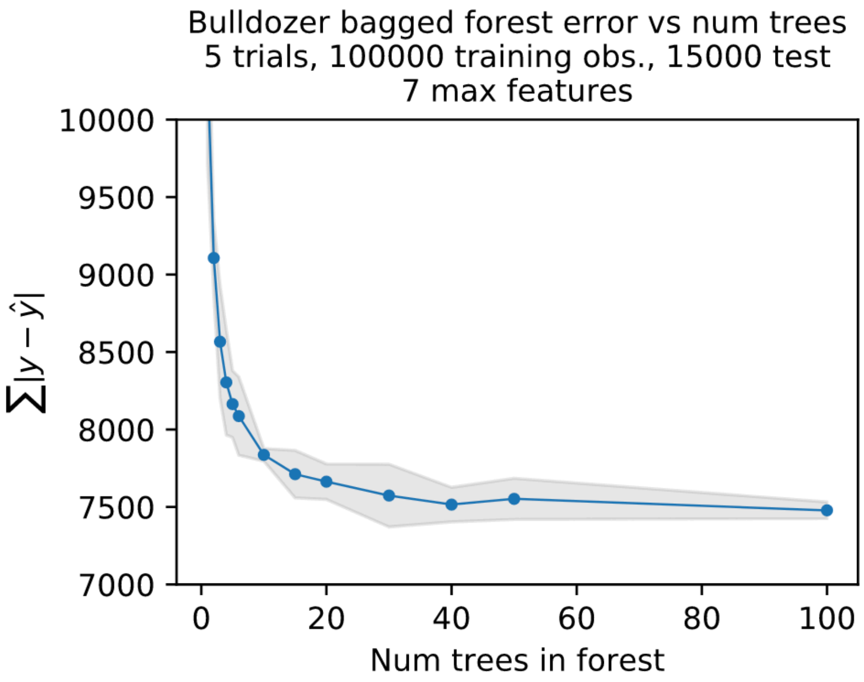
If max_features too low, bad accuracy



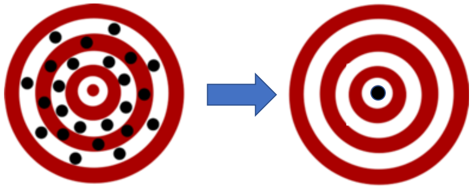


Effect of forest size on accuracy

- Why does accuracy improve greatly (initially) as we add trees?
  - Each tree sees only 2/3 of data so adding bootstrapped trees increases use of training data
- Why does accuracy asymptotically approach a minimum instead of continual improvement?
  - With enough trees, ensemble sees 100% of the training data; it's approaching the bias of single decision tree in ideal world



Properties (see *Breiman 2001*)



- p4 "*Random forests do not overfit as more trees are added*" **Why**?
  - Adding more trees actually REDUCES variance, and overfitting==variance
  - New trees get averaged in so each additional tree has less individual effect
  - New trees balance each other out, one might be too high, another too low
- p7 "*It's relatively robust to y outliers and X noise*" **Why**?
  - y outliers get shunted to their own leaf since doing so reducing loss function, particularly if squared-error is used
  - Noise vars aren't predictive so not chosen as split vars
- p10 Bagging helps more, the more unstable the model. **Why**?
  - Averaging is a smoothing operator, squeezing predictions to centroid
  - If model is low variance already, there is no point in bagging

Properties continued

- RFs are scale and range insensitive in features and target y **Why**?
  - Comparing feature values in decision nodes, not doing math on them
  - Computing mean or mode of y to predict
- ESLII p596 “*Classifiers are less sensitive to variance [than regressors]*” **Why**?
  - (not sure haha) I believe it has something to do with mode vs mean (mode is same until a threshold whereas mean is influenced by any value added, unless it is also the mean)

Bootstrapping vs subsampling

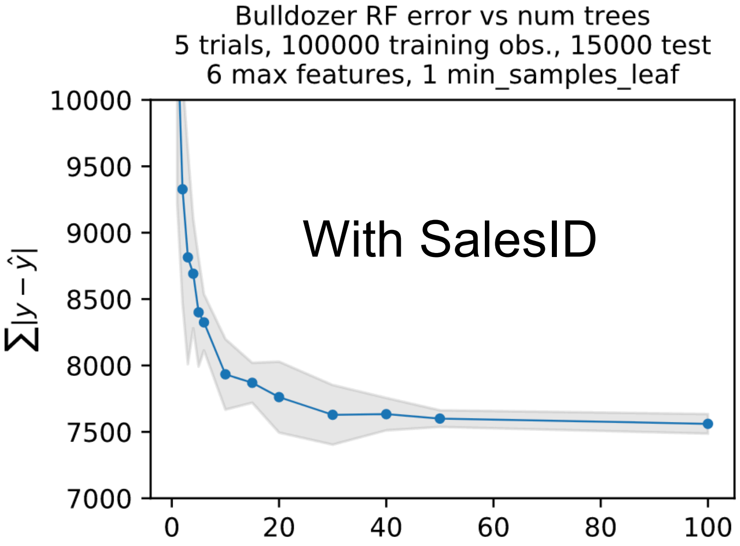
- Bootstrapping is sampling with replacement vs subsampling w/o replacement
- 😲 Friedman and Hall (2000): subsampling also works, showing that training trees with $n/2$ subsamples is similar in bias/variance to bagging http://statweb.stanford.edu/~jhf/ftp/bag.pdf
- Smaller training set is a big win in terms of speed
- Using even smaller fractions of n improve generality (reduce variance) because trees are less correlated (they work on different data chunks); note that each tree would become more biased as n subsample size decreases

RF Tuning strategy

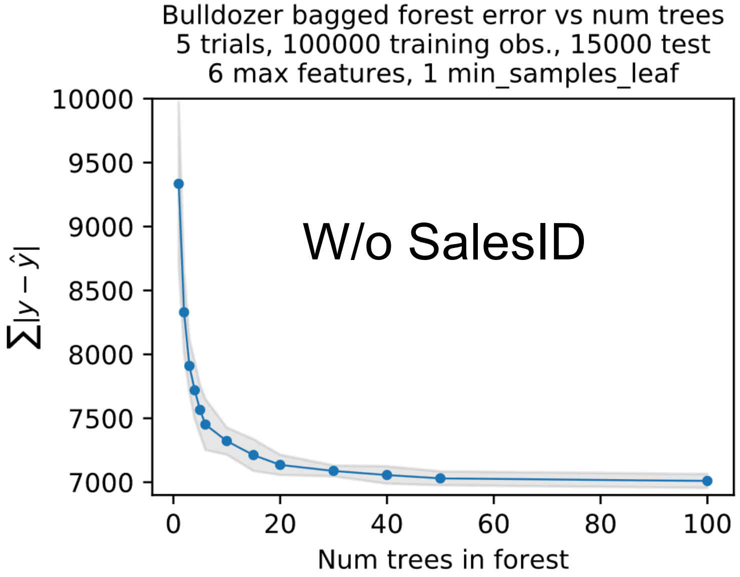
- Good news: very little tuning needed
- Start with maybe 20 trees and work upwards til validation error stops getting better; or just pick 100
- Sklearn uses max_features= sqrt(p) by default; try dropping this to log(p), or similar; ESLII suggests p/3 for regression and sqrt(p) for classification
- Try adjusting min samples per leaf: 1, 3, 5, 10, 25, 100
- Goal: minimize validation error
- Can also try grid search, but I never bother; Start with num trees, then tune the others

Feature engineering beats model tuning

- SalesID: unique record ID, and is never seen again in future predictions
- Is that useful for prediction? No
- Does the model think it’s useful? Yes
- Model is overfit not on noise but on falsely-predictive feature
  - Could be that sales ID correlates with inflation or change in type of models sold in auction creates “trend” in sale prices
- A case where using LESS data improves the model a lot ($500 diff)
- Dropping useless features also often gives a small bump



Bulldozer RF error vs num trees
5 trials, 100000 training obs., 15000 test
6 max features, 1 min_samples_leaf



Bulldozer bagged forest error vs num trees
5 trials, 100000 training obs., 15000 test
6 max features, 1 min_samples_leaf

Bagging summary

- Adding bagged trees does not increase **bias**; the avg of tree predictions is same as the expected prediction from any single bootstrapped tree (they are from same distribution: i.d.)
- Keep in mind, however, that a tree fit to bootstrapped data is only using 2/3 of the data and so each bagged tree will be less accurate than a single decision tree fit to entire data set
- Bagging does reduce **variance**, it is averaging predictions from lots of non-identical models afterall
- …

The RF algorithms

Fitting RFs

```
**Algorithm:** $RFfit(X,\ y,\ loss,\ ntrees,\ max\_features,\ min\_samples\_leaf)$
  **for** $i = 1..ntrees$ **do**
    $X', y' = bootstrap(X, y, size = |X|)$
    $T_i = RFdtreefit(X', y', loss, max\_features, min\_samples\_leaf)$
  **end**
```

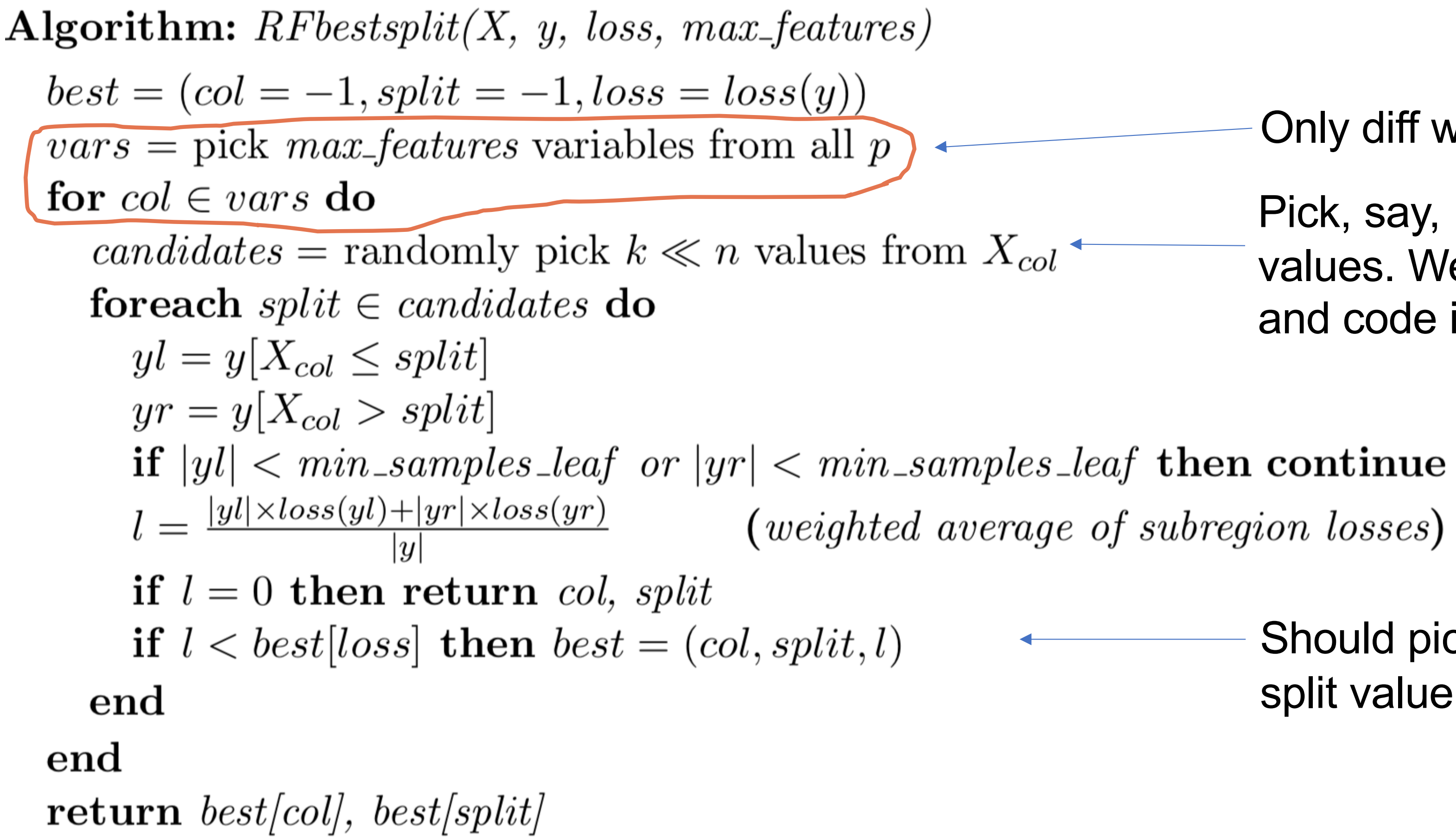
For regression, pass in loss = MSE or stddev
For classifier, pass in loss = gini

Fitting a single tree in RF

Same as decision tree except we pass max_features to RFbestsplit()

```
**Algorithm:** $RFdtreefit(X,\ y,\ loss,\ max\_features,\ min\_samples\_leaf)$
    **if** $|X| \leq min\_samples\_leaf$ **then** return Leaf($y$)
    $col, split = RFbestsplit(X, y, loss, max\_features)$
    **if** $col$ = $-1$ **then** return Leaf($y$)
    $lchild = RFdtreefit(X[X_{col} \leq split], y[X_{col} \leq split], ...)$
    $rchild = RFdtreefit(X[X_{col} > split], y[X_{col} > split], ...)$
    **return** $DecisionNode(col, split, lchild, rchild)$
```

Finding best split in decision node in RF



```
Algorithm: $RFbestsplit(X,\ y,\ loss,\ max\_features)$
  $best = (col = -1, split = -1, loss = loss(y))$
  $vars$ = pick $max\_features$ variables from all $p$
  for $col \in vars$ do
    $candidates$ = randomly pick $k \ll n$ values from $X_{col}$
    foreach $split \in candidates$ do
      $yl = y[X_{col} \leq split]$
      $yr = y[X_{col} > split]$
      if $|yl| < min\_samples\_leaf$ or $|yr| < min\_samples\_leaf$ then continue
      $l = \frac{|yl| \times loss(yl) + |yr| \times loss(yr)}{|y|}$          $(weighted\ average\ of\ subregion\ losses)$
      if $l = 0$ then return $col,\ split$
      if $l < best[loss]$ then $best = (col, split, l)$
    end
  end
  return $best[col],\ best[split]$
```

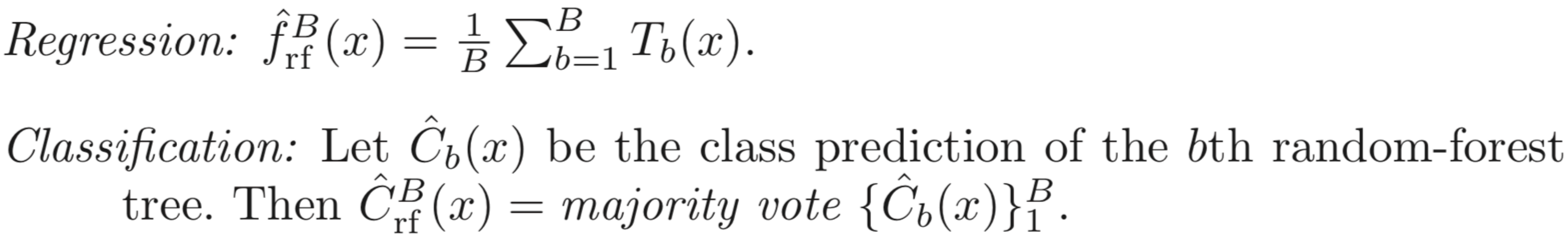
Only diff with decision tree

Pick, say, 11 not all possible X values. We get better generality and code is much faster!

Should pick midpoint between split value and next smallest X

Simplest RF prediction (ESLII p588)

- But doesn’t use all information to make best prediction
- Should use weighted averages / votes



Regression: $\hat{f}^B_{\mathrm{rf}}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}^B_{\mathrm{rf}}(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

RF prediction

Weighted average of y values among the leaves reached by running x down each tree

```
**Algorithm:** $RFpredict_{regr}(\{T_1..T_{ntrees}\},\ x)$
    **Let** $leaves = \{leaf(T_t, x)\ \forall\, t = 1..ntrees\}$
    $nobs = \sum_{t=1}^{ntrees} |leaves_t|$
    $ysum = \sum_{t=1}^{ntrees} \sum_{y \in leaves_t} y$
    **return** $\frac{1}{nobs} ysum$
```

Count all y votes among the leaves reached by running x down each tree

```
**Algorithm:** $RFpredict_{class}(\{T_1..T_{ntrees}\},\ x)$
    $counts[k] = 0\ \forall$ classes $k$
    **foreach** $t = 1..ntrees$ **do**
        $leaf\ = leaf(T_t, x)$            $(leaf\ reached\ by\ x)$
        **foreach** $y \in leaf$ **do**
            $counts[y]$ += $1$            $(track\ count\ of\ leaf\ modes)$
        **end**
    **end**
    **return** argmax$(counts)$
```