COMS 4721: Machine Learning for Data Science
Lecture 12, 2/28/2017

Prof. John Paisley

Department of Electrical Engineering
& Data Science Institute

Columbia University

DECISION TREES

(not linear classification technique)

DECISION TREES

real value or class

A *decision tree* maps input $x \in \mathbb{R}^d$ to output y using binary decision rules:

- Each node in the tree has a *splitting rule*.
- Each leaf node is associated with an output value (outputs can repeat).

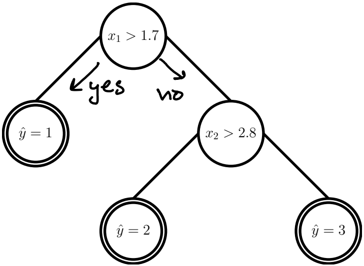
Each splitting rule is of the form

$$h(x) = \mathbb{1}\{x_j > t\}$$

for some dimension j of x and $t \in \mathbb{R}$.

Using these transition rules, a path to a *leaf node* gives the prediction.

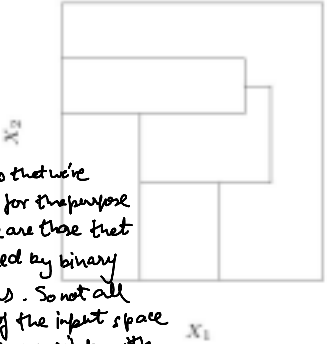
(One-level tree = *decision stump*)



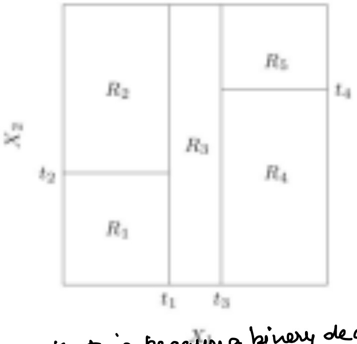
Regression Trees

→ Why?

This type of region is not something we can get with a binary decision tree.



The partitions that we're interested in for the purpose of this lecture are those that can be defined by binary splitting trees. So not all partitions of the input space are going to be possible with the technique we're going to discuss.



And the reason for that is because a binary decision tree is something that's very easy to learn.

Motivation: Partition the space so that data in a region have same prediction

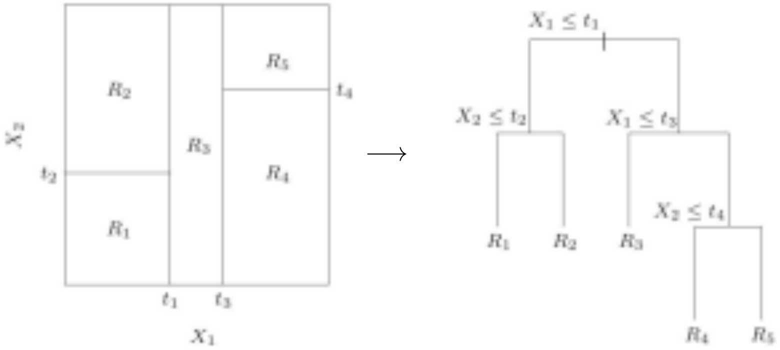
Left: Difficult to define a "rule".

Right: Easy to define a recursive splitting rule.

So for regression / classification if a data point falls within a particular region. Any part of that region is going to have the same exact prediction.

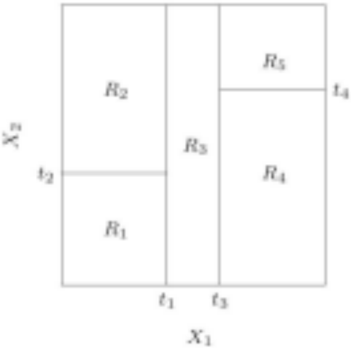
Regression Trees

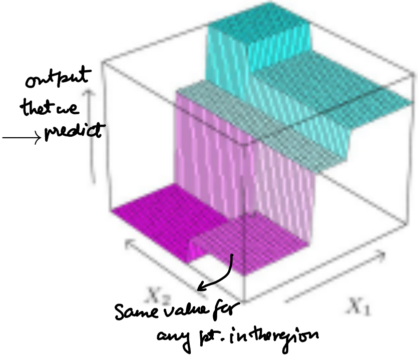
If we have a new i/p x, and we want to say which of the 5 decision regions does x fall in. We can quickly find it according to this decision tree.



If we think in terms of trees, we can define a simple rule for partitioning the space. The left and right figures represent the same regression function.

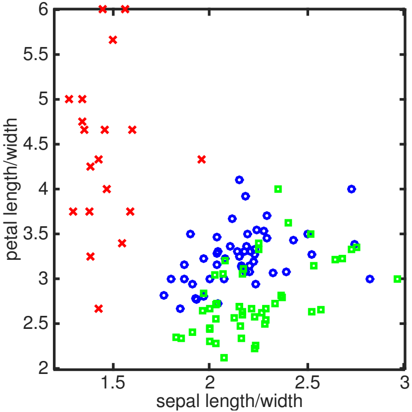
REGRESSION TREES





Adding an output dimension to the figure (right), we can see how regression trees can learn a step function approximation to the data.

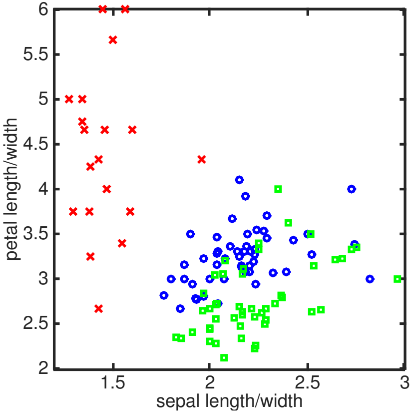
Classification Trees (Example)



Classifying irises using sepal and petal measurements:

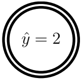
- $x \in \mathbb{R}^2$, $y \in \{1, 2, 3\}$
- x_1 = ratio of sepal length to width
- x_2 = ratio of petal length to width

Classification Trees (Example)

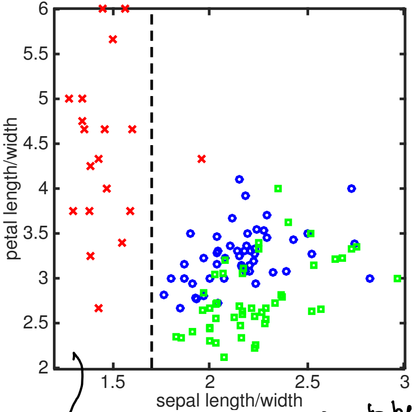


Classifying irises using sepal and petal measurements:

- $x \in \mathbb{R}^2$, $y \in \{1, 2, 3\}$
- x_1 = ratio of sepal length to width
- x_2 = ratio of petal length to width



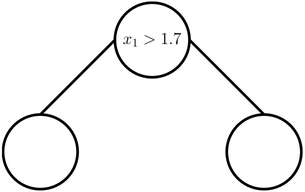
Classification Trees (Example)



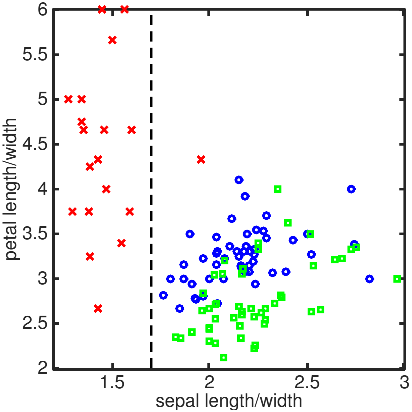
simply make the prediction to be the most prevalent class in that region.

Classifying irises using sepal and petal measurements:

- $x \in \mathbb{R}^2$, $y \in \{1, 2, 3\}$
- x_1 = ratio of sepal length to width
- x_2 = ratio of petal length to width

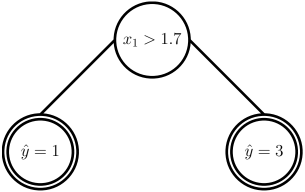


Classification Trees (Example)

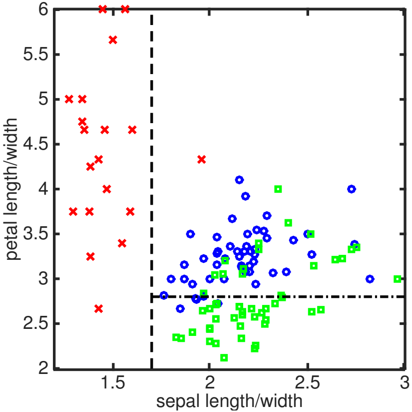


Classifying irises using sepal and petal measurements:

- $x \in \mathbb{R}^2$, $y \in \{1, 2, 3\}$
- x_1 = ratio of sepal length to width
- x_2 = ratio of petal length to width

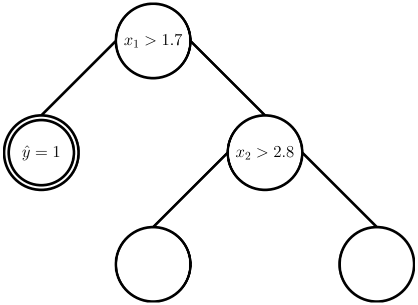


Classification Trees (Example)

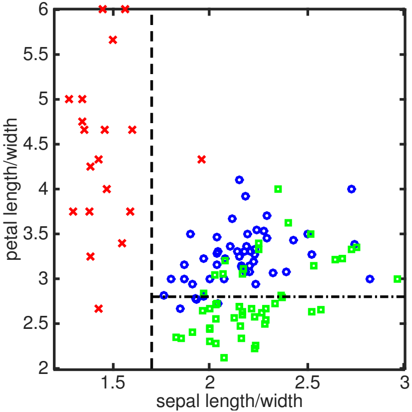


Classifying irises using sepal and petal measurements:

- $x \in \mathbb{R}^2$, $y \in \{1, 2, 3\}$
- x_1 = ratio of sepal length to width
- x_2 = ratio of petal length to width

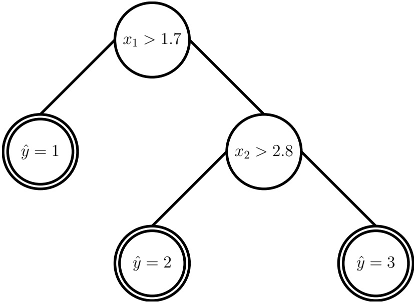


Classification Trees (Example)



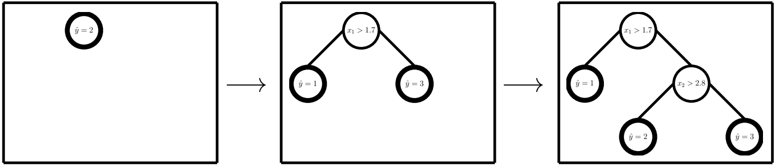
Classifying irises using sepal and petal measurements:

- $x \in \mathbb{R}^2$, $y \in \{1, 2, 3\}$
- x_1 = ratio of sepal length to width
- x_2 = ratio of petal length to width



BASIC DECISION TREE LEARNING ALGORITHM

(come up with a way to learn a decision tree from data)



The basic method for learning trees is with a top-down greedy algorithm.

- Start with a single leaf node containing all data (Basically follow the same sequence of events from previous slides.)
- Loop through the following steps:
  - Pick the leaf to split that reduces uncertainty the most.
  - Figure out the $\lessgtr$ decision rule on one of the dimensions.

  (So for example, we would evaluate both leaves, see which one is the best to split and we would make the split.)
- Stopping rule discussed later.

So we grow the tree from nothing to something.

Label/response of the leaf is majority-vote/average of data assigned to it.

Growing a regression tree

max. decision regions: n-1 ?

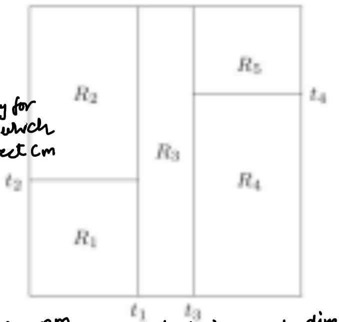
How do we grow a regression tree?

- For M regions of the space, $R_1, \ldots, R_M$, the prediction function is

this indicator will be true only for the region in which x falls, which will then pick out the correct c_m

$$f(x) = \sum_{m=1}^{M} c_m \mathbb{1}\{x \in R_m\}.$$

So for a fixed M, we need R_m and c_m.



Goal: Try to minimize $\sum_i (y_i - f(x_i))^2$

We look at all of our data, we say which of these points fall region R^m. We then take their corresponding outputs y in the training set, and then average over those outputs. So that's how we can minimize the sum of squared errors.

1. Find c_m given R_m: Simply the average of all y_i for which $x_i \in R_m$.

really just a brute force method.

2. How do we find regions? Consider splitting region R at value s of dim j: *
   - Define $R^-(j,s) = \{x_i \in R | x_i(j) \leq s\}$ and $R^+(j,s) = \{x_i \in R | x_i(j) > s\}$
   - For each dimension j, calculate the best splitting point s for that dimension.
   - Do this for each region (leaf node). Pick the one that reduces the objective most.

*
We look individually at each region and decide what's the benefit we can get splitting along that.
For ex: we take R_1, and evaluate all possible points at which we can split it along each dimension and say what would be the result, how much would I reduce the objective function (sum of squared errors) if I did that.

** We have then a proposed dimension j and proposed value s for a particular region r. We then say how would this partition the data in region r.?

*** So we construct a set R^{-}, which all of the region in R such that j^{th} dimension is less than S.

**** We then update the proposed predictions. The proposed value for C if we made that split and we look at how much we reduce the resulting objective.

So we do this for every single region, we do it for all possible splitting points s. that would lead to unique subsets, a unique partition. And we can for each of those proposals get a new possible regression F and we evaluate each of them.

Growing a classification tree

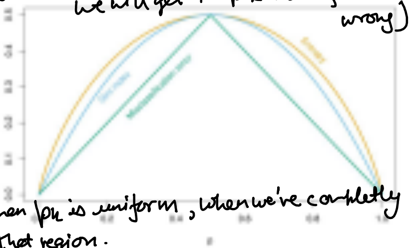
For regression: Squared error is a natural way to define the splitting rule.

For classification: Need some measure of how badly a region classifies data and how much it can improve if it's split.

K-class problem: For all $x \in R_m$, let p_k be empirical fraction labeled k.

most prevalent label (we won't that one wrong but we will get $1-p_k$ that fraction wrong)

Measures of quality of R_m include



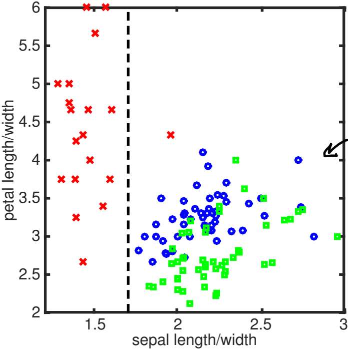
1. Classification error: $1 - \max_k p_k$
2. Gini index: $1 - \sum_k p_k^2$
3. Entropy: $-\sum_k p_k \ln p_k$

each of these is going to be maximized when p_k is uniform, when we're completely uncertain about what the label could be for that region.

- These are all *maximized* when p_k is uniform on the K classes in R_m.
- These are *minimized* when $p_k = 1$ for some k (R_m only contains one class)

[all of data in a region just falls into 1 particular class]

Growing a classification tree



Search R_1 and R_2 for splitting options.

1. R_1: $y = 1$ leaf classifies perfectly
2. R_2: $y = 3$ leaf has Gini index

101 – Total points.

$$u(R_2) = 1 - \left(\frac{1}{101}\right)^2 - \left(\frac{50}{101}\right)^2 - \left(\frac{50}{101}\right)^2$$

red green blue

$$= 0.5098$$

We now want pick a dimension to split and a point to split to minimize the resulting genie coefficient of the 2 regions.

Gini improvement from split R_m to R_m^- & R_m^+:

genie coefficient for the resulting data that fall in those 2 regions.

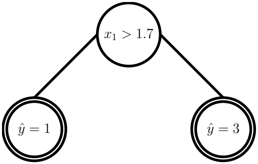
$$u(R_m) - \left(p_{R_m^-} \cdot u(R_m^-) + p_{R_m^+} \cdot u(R_m^+)\right)$$

Original region

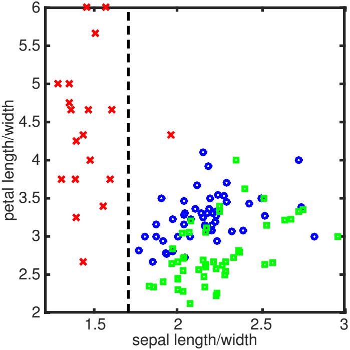
fraction of data that fall in those regions.

$p_{R_m^+}$: Fraction of data in R_m split into R_m^+. 2 regions

$u(R_m^+)$: New quality measure in region R_m^+.



GROWING A CLASSIFICATION TREE

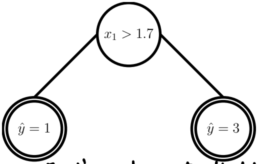


Search R_1 and R_2 for splitting options.

1. R_1: $y = 1$ leaf classifies perfectly
2. R_2: $y = 3$ leaf has Gini index

$$u(R_2) = 1 - \left(\frac{1}{101}\right)^2 - \left(\frac{50}{101}\right)^2 - \left(\frac{50}{101}\right)^2$$
$$= 0.5098$$

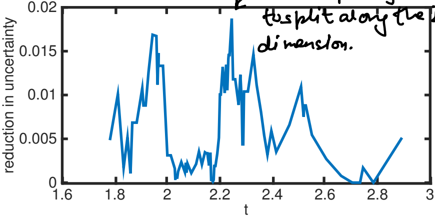
At a particular split we get a reduction value,



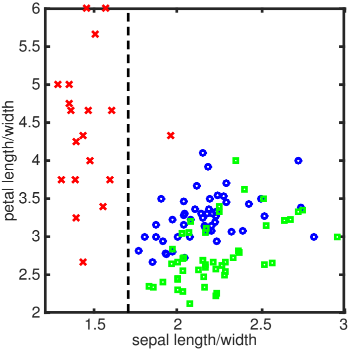
Let's check both dimensions at all of the possible points for a particular region

Check split R_2 with $\mathbb{1}\{x_1 > t\}$

best split if we were to split along the 1st dimension.



Growing a classification tree



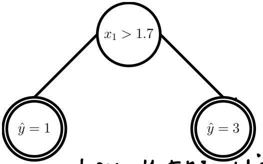
Search R_1 and R_2 for splitting options.

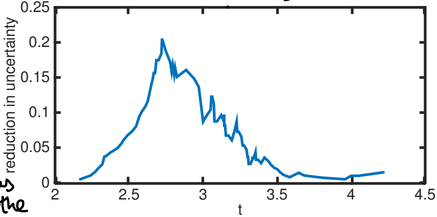
1. R_1: $y = 1$ leaf classifies perfectly
2. R_2: $y = 3$ leaf has Gini index

$$u(R_2) = 1 - \left(\frac{1}{101}\right)^2 - \left(\frac{50}{101}\right)^2 - \left(\frac{50}{101}\right)^2$$
$$= 0.5098$$

So the reduction we're getting by looking

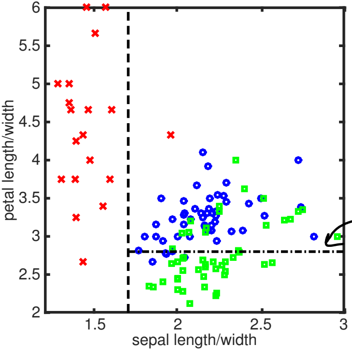
Check split R_2 with $\mathbb{1}\{x_2 > t\}$ in the II^{nd} dimension is massive compared to the reduction from previous one.





That intuitively just says that this split makes us much more confident about the 2 classes in each of the resulting regions.

GROWING A CLASSIFICATION TREE



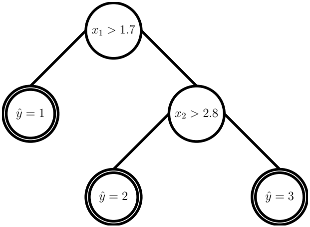
Search R_1 and R_2 for splitting options.

1. R_1: $y = 1$ leaf classifies perfectly
2. R_2: $y = 3$ leaf has Gini index

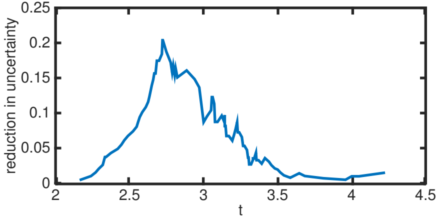
$$u(R_2) = 1 - \left(\frac{1}{101}\right)^2 - \left(\frac{50}{101}\right)^2 - \left(\frac{50}{101}\right)^2$$

$$= 0.5098$$

we decide to split here.



Check split R_2 with $\mathbb{1}\{x_2 > t\}$



PRUNING A TREE

(splitting region)

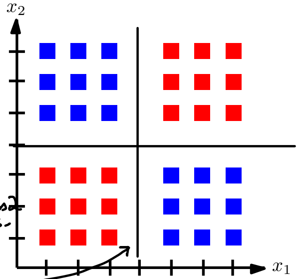
Q: When should we stop growing a tree?

A: Uncertainty reduction is not best way.

Example: Any split of x_1 or x_2 at right will show *zero* reduction in uncertainty.

All splits that we can make, we have the same amount of class 1 and class 2 in both splits.

However, we can learn a perfect tree on this data by partitioning in quadrants.



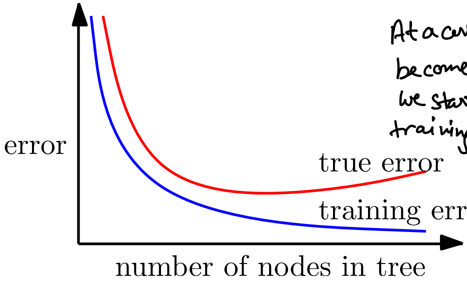
Pruning is the method most often used. Grow the tree to a very large size. Then use an algorithm to trim it back.

(prune the tree) We make many more splits than are possibly necessary.

(We won't cover the algorithm, but mention that it's non-trivial.)

OVERFITTING

If we grow a tree out too big, we can eventually partition the input space into enough regions where we perfectly classify / predict everything.



At a certain point, our trees become too complex and we start to overfit to the training data and we make worse test predictions because it doesn't generalize the new data as well.

- *Training error* goes to zero as size of tree increases.
- *Testing error* decreases, but then increases because of *overfitting*.

THE BOOTSTRAP

(general technique – that's used for much more than just trees.)

The Bootstrap: A Resampling Technique

We briefly present a technique called the *bootstrap*. This statistical technique is used as the basis for learning *ensemble classifiers*.

where we essentially resample from our data. And then we use these resampled data to improve some sort of an estimator.

Bootstrap

Bootstrap (i.e., resampling) is a technique for improving estimators.

Resampling = Sampling from the empirical distribution of the data

Application to ensemble methods

- We will use resampling to generate many "mediocre" classifiers, that are not so great.

However, when we put them together using a technique called bagging, their combination suddenly makes the overall classification much better.

- We then discuss how "bagging" these classifiers improves performance.
- First, we cover the bootstrap in a simpler context.

Bootstrap: Basic algorithm

we might be interested in learning the median of the true underlying distribution that generated this data. So we don't care about the median of the actual data we have. we want the median of whatever distribution generated this data that we don't get to see.

Input

- A sample of data $x_1, \ldots, x_n$.
- An estimation rule $\hat{S}$ of a statistic S. For example, $\hat{S} = \text{med}(x_{1:n})$ estimates the true median S of the unknown distribution on x.

Bootstrap algorithm

1. Generate *bootstrap samples* $\mathcal{B}_1, \ldots, \mathcal{B}_B$. [we have B different bootstrap samples.]
   - Create $\mathcal{B}_b$ by picking points from $\{x_1, \ldots, x_n\}$ randomly n times (with replacement)
   - A particular x_i can appear in $\mathcal{B}_b$ many times (it's simply duplicated).
2. Evaluate the estimator on each $\mathcal{B}_b$ by pretending it's the data set:

   When we have to calculate our statistic, we simply calculate it using the bootstrap set.

   $$\hat{S}_b := \hat{S}(\mathcal{B}_b)$$

   [Each of these bootstrap datasets is like a view of our underlying original dataset.]
3. Estimate the mean and variance of $\hat{S}$: (by averaging over the bootstrapped calculations of this statistic.)

   $$\mu_B = \frac{1}{B}\sum_{b=1}^{B} \hat{S}_b, \quad \sigma_B^2 = \frac{1}{B}\sum_{b=1}^{B} (\hat{S}_b - \mu_B)^2$$

   gives us some uncertainty

* Ofcourse, because we don't have an infinite number of samples from the underlying distribution, we can't find out what exactly is the median. So what we do is we have an estimator $\hat{S}$ which estimates the true underlying median to be the median of our dataset.

Example: Variance estimation of the median

That's our estimate of the median of the underlying distribution that generated this dataset.
So the issue is how confident can we be in that estimation?

- The median of $x_1, \ldots, x_n$ (for $x \in \mathbb{R}$) is found by simply sorting them and taking the middle one, or the average of the two middle ones.

- How confident can we be in the estimate median$(x_1, \ldots, x_n)$?
  - Find it's variance.
  - But how? Answer: By bootstrapping the data.

we get only possible value for the prediction of the median. How can we get multiple values or somehow evaluate our uncertainty about that prediction.

1. Generate bootstrap data sets $\mathcal{B}_1, \ldots, \mathcal{B}_B$.

Empirical variance of the median that we get from each of these individual datasets.

2. Calculate: (notice that $\hat{S}_{mean}$ is the mean of the median)

$$\hat{S}_{mean} = \frac{1}{B}\sum_{b=1}^{B} \text{median}(\mathcal{B}_b), \quad \hat{S}_{var} = \frac{1}{B}\sum_{b=1}^{B}\Big(\text{median}(\mathcal{B}_b) - \hat{S}_{mean}\Big)^2$$

- The procedure is remarkably simple, but has a lot of theory behind it.

Our expected median each individual bootstrap dataset. that median that we are going to predict has an expectation of

So that's a way to gauge our uncertainity about the true underlying median.

Bagging and Random Forests

BAGGING

Bagging uses the bootstrap for regression or classification:

$$\textbf{Bagging} = \textbf{B}\text{ootstrap } \textbf{agg}\text{regation}$$

[So we bootstrap the dataset, we learn some model on each bootstrap set and then we aggregate the results to get one final prediction.]

Algorithm

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

going to have repeats of certain data points and some other data points are not going to be in that dataset.

1. Draw a bootstrap sample $\mathcal{B}_b$ of size n from training data.
2. Train a classifier or regression model f_b on $\mathcal{B}_b$.

[We have B of these classifiers/regression models, each one is learned on a particular bootstrap dataset.]

- For a new point x_0, compute:

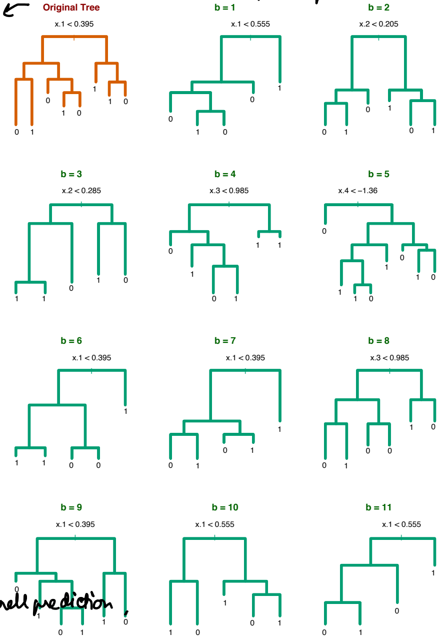
$$f_{\text{avg}}(x_0) = \frac{1}{B} \sum_{b=1}^{B} f_b(x_0)$$

  - For regression, $f_{\text{avg}}(x_0)$ is the prediction.
  - For classification, view $f_{\text{avg}}(x_0)$ as an average over B votes. Pick the majority.

EXAMPLE: BAGGING TREES (When we want to bag trees)

resample n times.

learn on original dataset.



- Binary classification, $x \in \mathbb{R}^5$.
- Note the variation among bootstrapped trees.
- Take-home message:

  With bagging, each tree doesn't have to be great, just "ok".
- Bagging often improves results *when the function is non-linear.*

So these are 11 different views of our dataset which will give us 11 different trees. Each of these trees doesn't have to be great but their combination is suddenly going to improve our overall prediction.

So we can see that when we go from 1 tree up to say 25 trees, using these bootstrap samples, we get a clear improvement. But then at a certain point the improvement stops.

Drawbacks of Bagging how?

- Bagging works on trees because of the bias-variance tradeoff (↑ bias, ↓ variance).

There's a certain limit to the performance, And in general because each of the trees that we learn is highly correlated with the previous one. And that's why we suddenly stop improving.

- However, the bagged trees are correlated.
- In general, when bootstrap samples are correlated, the benefit of bagging decreases.

Random Forests → very simple modification of bagging trees.

Modification of bagging where trees are designed to reduce correlation. of bagging trees.

- A very simple modification.
- Still learn a tree on each bootstrap set, $\mathcal{B}_b$.
- To split a region, only consider random subset of dimensions of $x \in \mathbb{R}^d$.

↳ and only make a partition along one of these dimensions.

Random Forests: Algorithm

number of dimensions that we're going to consider on each split.

Training

Input parameter: m — a positive integer with $m < d$, often $m \approx \sqrt{d}$

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

1. Draw a bootstrap sample $\mathcal{B}_b$ of size n from the training data.
2. Train a tree classifier on $\mathcal{B}_b$, where each split is computed as follows:
   or all splits of a tree.
   - Randomly select m dimensions of $x \in \mathbb{R}^d$, newly chosen for each b.
   - Make the best split restricted to that subset of dimensions. ?

Each tree is only going to consider a small subset of all our dimensions.

- Bagging for trees: Bag trees learned using the original algorithm.
- Random forests: Bag trees learned using algorithm on this slide.

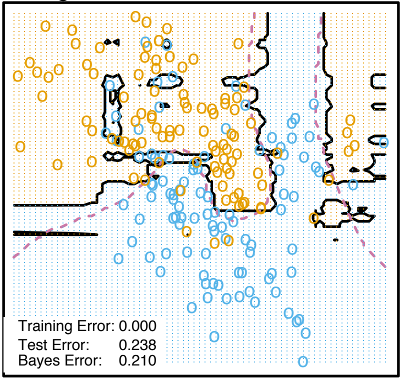
Each tree is only learnt only on a random subset of the originial Dolimensions. What is going to do is breakdown the correlations between the trees that we learned.

Random Forests

Notice that we can learn some quite complicated decision boundaries (However, because our trees are splitting along the 2 dimensions, they tend to align the decision boundaries with the 2 dimensions.)

Example problem

- Random forest classification.
- Forest size: A few hundred trees.
- Notice there is a tendency to align decision boundary with the axis.



Let each tree make it's prediction, and then we take a majority vote of these few hundred trees.