Big Data BUS 41201

Week 9: Trees

Veronika Ročková

University of Chicago Booth School of Business

http://faculty.chicagobooth.edu/veronika.rockova/

- ✓ Using tree-logic to make predictions.

- ✓ **C**lassification **A**nd **R**egression **T**rees.

- ✓ **CART:** Greedy growing and tree pruning.

- ✓ Trees in `R`: library `trees`.

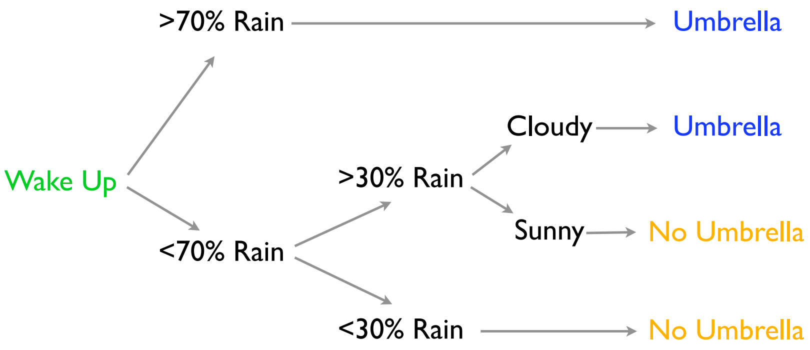
- ✓ Bagging and Random Forests.

- ✓ Random forests in `R`: library `rforests`.

- ✓ *Simple examples*: NBC, prostate cancer, motorcycle crashes.

- ✓ *Larger example* on house prices in California.

What is a Decision Tree?



Tree-logic uses a sequence of inquiries to come to a conclusion.

The trick is to have mini-decisions combine for good choices.

Each decision is a node, and the final prediction is a leaf node

Tree-based Statistical Learning

Tree-based learning: predicting outcome y from predictors $\mathbf{x} = (x_1, \ldots, x_p)'$ by dividing up the feature space into small regions where the outcomes are more similar.

Within each region, a very simple model in fit locally.

This works both when y is *categorical* and *continuous*, i.e., both for classification and regression

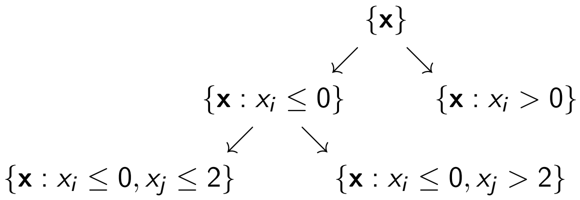
Regions can be achieved by making successive binary splits on the predictors variables $x_1, \ldots x_p$, i.e. we choose a variable x_j, $j = 1, \ldots p$, divide up the feature space according to

$$x_j \leq c \quad \text{and} \quad x_j > c$$

Then we repeat the same on each half.

Decision trees are like a game of mousetrap

You drop your **x** covariates in at the top, and each decision node bounces you either left or right. Finally, you end up in a leaf node which contains the data subset defined by these decisions (splits).



$\{\mathbf{x}\}$

$\{\mathbf{x} : x_i \leq 0\}$ $\{\mathbf{x} : x_i > 0\}$

$\{\mathbf{x} : x_i \leq 0, x_j \leq 2\}$ $\{\mathbf{x} : x_i \leq 0, x_j > 2\}$

The prediction rule at each leaf (a class probability or predicted $\hat{y}$) is the average of the sample y values that ended up in that leaf.

Decision Trees are a Regression Model

You have inputs $\mathbf{x}$ (forecast, current conditions)
and an output of interest y (need for an umbrella).

Based on previous data, the goal is to specify branches of choices that lead to good predictions in new scenarios.
In other words, you want to estimate a Tree Model.

Instead of linear coefficients, we need to find 'decision nodes':
split-rules defined via thresholds on some dimension of $\mathbf{x}$.

Nodes have a parent-child structure: every node except the root has a parent, and every node except the leaves has two children.

Estimation of Decision Trees

As usual, we'll maximize data likelihood (minimize deviance).

But what are the observation probabilities in a tree model?

Two types of likelihood: **classification** and **regression** trees.

> A given covariate **x** dictates your path through tree nodes, leading to a leaf node at the end.

Classification trees have class probabilities at the leaves.

Probability I'll be in heavy rain is 0.9 (so take an umbrella).

Regression trees have a mean response at the leaves.

The expected amount of rain is 2in (so take an umbrella).

Tree deviance is the same as in linear models

Regression Deviance: $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

Classification Deviance: $-\sum_{i=1}^{n}\log(\hat{p}_{y_i})$

It is also common to use Gini Deviance $\sum_{i=1}^{n}\hat{p}_{y_i}(1-\hat{p}_{y_i})$

Instead of being based on $\mathbf{x}'\boldsymbol{\beta}$, predicted $\hat{p}$ and $\hat{y}$ are functions of $\mathbf{x}$ passed through the decision nodes.

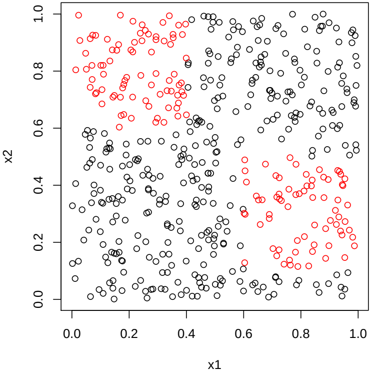
We need a way to estimate the sequence of decisions.

- How many are they? What is the order?

☹ There is a *huge* set of possible tree configurations.

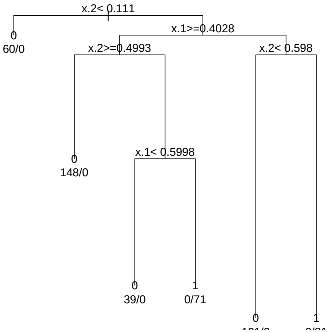
Toy Example: Classification Tree

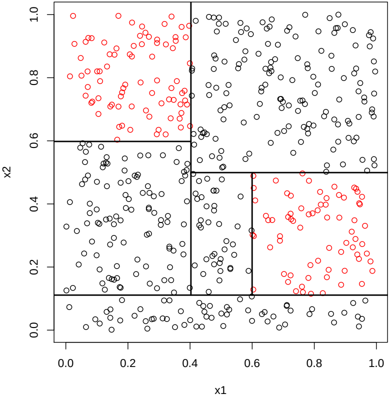
Example: $n = 500$ points in $p = 2$ dimensions, falling into classes 0 and 1, as marked by colors



Does dividing up the feature space into rectangles look like it would work here?

Toy Example: Classification Tree





Classification Trees

Classification trees are very popular because they are *interpretable* (they mimic how decisions are made).

In classification, $y_i \in \{1, \dots K\}$ are the class labels, and $x_i \in \mathbb{R}^p$ measure the p predictor variables.

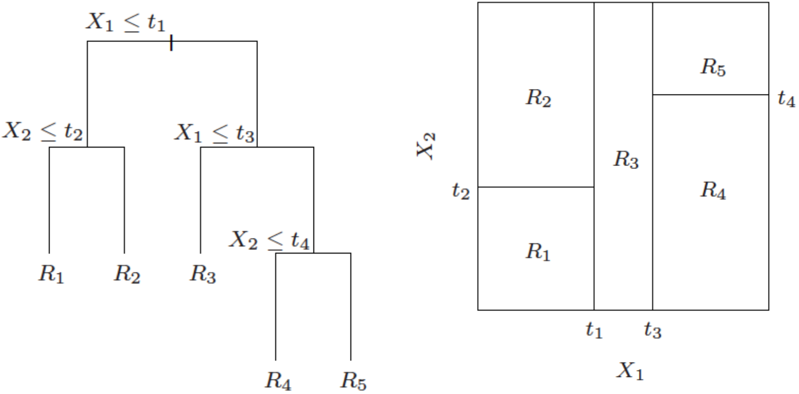
The classification tree can be thought of as defining m regions (rectangles) $R_1, \dots R_m$, each corresponding to a leaf of the tree

We assign each R_j a class label $c_j \in \{1, \dots K\}$ (typically the most dominant class within the region).

We then classify a new point $\mathbf{x}$ as c_j if it falls in region R_j.

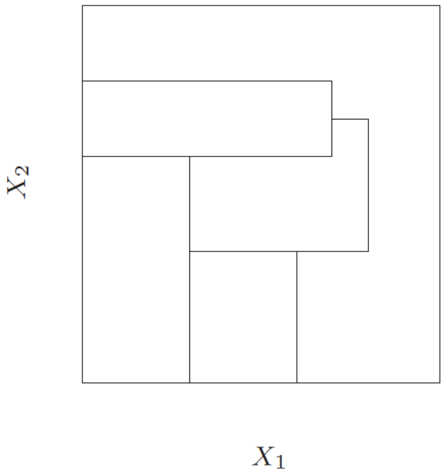
☺ *Finding out which region a given point x belongs to is easy since the regions R_j are defined by a tree.*

Example: regions defined by a tree



(From ESL page 306)

Example: other regions



(From ESL page 306)

Predicted Class Probabilities

☺ With classification trees, we get not only the predicted classes for new points but also the *predicted class probabilities*.

Note that each region R_j contains *some subset of the training data* (x_i, y_i), say, **$\mathbf{n_j}$ points**.

Further, for each class $k = 1, \dots K$, we can estimate the probability that the class label is k *given that the feature vector lies in region R_j*

$$\widehat{\mathbb{P}}(C = k | X \in R_j) = \frac{\# y_i \, : \, y_i \in R_j \quad \text{and} \quad y_i = k}{\# y_i \, : \, y_i \in R_j}$$

the proportion of points in the region that are of class k.

The predicted class $\widehat{c_j}$ is *the most common occurring class* among these points $\widehat{c_j} = \arg\max_{k=1,\dots K} \; \widehat{\mathbb{P}}(C = k | X \in R_j)$

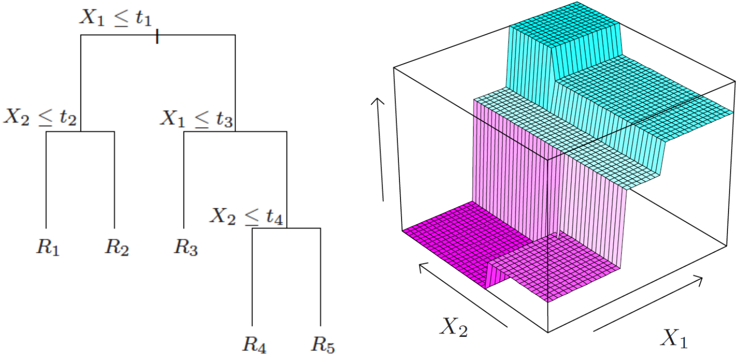
Classification Trees and Their Competitors

	Model assumptions?	Estimated probabilities?	Interpretable?	Flexible?
LDA	Yes	Yes	Yes	No
LR	Yes	Yes	Yes	No
k-NN	No	No	No	Yes
Trees	No	Yes	Yes	Somewhat

	Predicts well?
LDA	Depends on X
LR	Depends on X
k-NN	If properly tuned
Trees	?

Regression Trees

Suppose that now we want to predict a *continuous* outcome instead of a class label. Essentially, everything follows as before, but now we just fit a mean of a continuous outcome rather than a proportion inside each rectangle



Regression Trees

The estimated regression function has the form

$$\mathbb{E}[y|x] \ = \ \sum_{j=1}^{m} c_j \cdot \mathbb{I}\{x \in R_j\} \ = \ c_j \text{ such that } x \in R_j$$

just as it did with classification. The quantities c_j are no longer predicted classes, but instead they are real numbers.

How would we choose c_j's?

☺ Simple: just take the average response of all of the points in the region,

$$c_j = \frac{1}{n_j} \sum_{x_i \in R_j} y_i$$

The main difference in building the tree is that we use sums of squares instead of misclassification error (or Gini index or deviance) to decide which region to split.

Trees: Recap

Given a parent set of data $\{\mathbf{x}_i, y_i\}_{i=1}^n$, the optimal split is that location x_{lj} on some dimension j on some observation i, so that the child sets

$$\text{left: } \{(\mathbf{x}_i, y_i) : \ x_{ij} \leq x_{lj}\} \ \text{ and } \ \text{right: } \{(\mathbf{x}_i, y_i) : \ x_{ij} > x_{lj}\}$$

are as homogeneous in response y as possible.

For example, we will minimize the sum of squared errors

$$\sum_{k \in \text{left}} (y_k - \bar{y}_{\text{left}})^2 + \sum_{k \in \text{right}} (y_k - \bar{y}_{\text{right}})^2$$

for *regression trees*, or gini impurity for *classification trees* (e.g., the sum across children 'c' of $n_{\text{c}}\bar{y}_{\text{c}}(1 - \bar{y}_{\text{c}})$ if $y \in \{0, 1\}$).

How to build trees?

There are two main issues to consider in building a tree:

1. *How to choose the splits?*
2. *How big to grow the tree?*

Think first about varying the depth of the tree ... which is more complex, a big tree or a small tree? What tradeoff is at play here? How might we eventually consider choosing the depth?

Now for a fixed depth, consider choosing the splits. If the tree has depth d, then it has $\approx 2^d$ nodes. At each node we could choose any of p the variables for the split—this means that the number of possibilities is

$$p \cdot 2^d$$

This is huge even for moderate d! And we haven't even counted the actual split points themselves

CART Recap

We estimate decision trees by being recursive and greedy

CART grows the tree through a sequence of splits:

- Given any set (node) of data, you can find the optimal split (the error minimizing split) and divide into two child sets.
- We then look at each child set, and again find the optimal split to divide it into two homogeneous subsets.
- The children become parents, and we look again for the optimal split on their new children (the grandchildren!).

You stop splitting and growing when the size of the leaf nodes hits some minimum threshold (e.g., say no less than 10 obsv per leaf). Often there are also minimum deviance improvement thresholds.

Use the tree library for CART in R

The syntax is essentially the same as for `glm`:

```
mytree = tree(y ~x1 + x2 + x3 ..., data=mydata)
```

There are only a few other possible arguments,
all of which dictate possible types of new children

- `mincut` is the minimum size for a new child.
- `mindev` is the minimum (proportion) deviance improvement for proceeding with a new split.

These are important: you may want to make them smaller than their defaults: `mincut=5`, `mindev=0.01`.

As usual, you can `print`, `summarize`, and `plot` the tree.

Recall our NBC data: viewer demographic % by show.

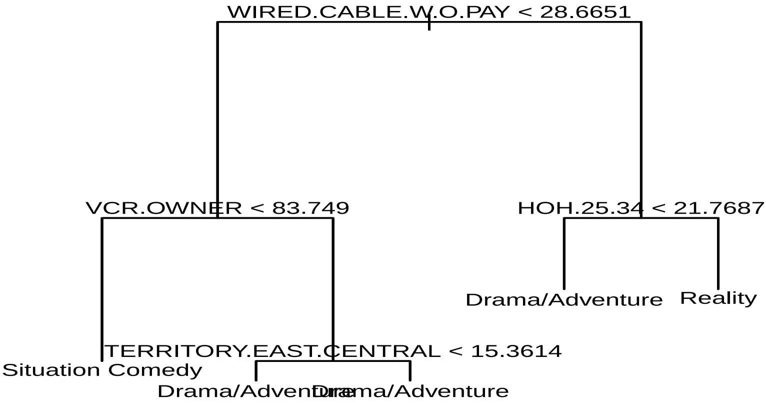
Consider a classification tree to predict genre from demographics.

```
> genretree
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

 1) root 40 75.800 Drama/Adventure ( 0.47500 0.42500 0.10000 )
   2) WIRED.CABLE.W.O.PAY < 28.6651 22 33.420 Drama/Adventure ( 0.72727 0.09091 0.1
     4) VCR.OWNER < 83.749 5  6.730 Situation Comedy ( 0.00000 0.40000 0.60000 ) *
     5) VCR.OWNER > 83.749 17  7.606 Drama/Adventure ( 0.94118 0.00000 0.05882 )
      10) TERRITORY.EAST.CENTRAL < 15.3614 12  0.000 Drama/Adventure ( 1.00000 0.00
      11) TERRITORY.EAST.CENTRAL > 15.3614 5  5.004 Drama/Adventure ( 0.80000 0.000
   3) WIRED.CABLE.W.O.PAY > 28.6651 18 16.220 Reality ( 0.16667 0.83333 0.00000 )
     6) HOH.25.34 < 21.7687 5  6.730 Drama/Adventure ( 0.60000 0.40000 0.00000 ) *
     7) HOH.25.34 > 21.7687 13  0.000 Reality ( 0.00000 1.00000 0.00000 ) *
```

Output from `tree` shows a series of decision nodes and the proportion in each genre at these nodes, down to the leaves.

Trees are easiest to understand in a dendrogram



Shows the sequence of internal splits, ending in leaf-node decisions.
Here, the decision is the genre of highest proportion in each leaf.
To get the dendrogram, do `plot(mytree)` then `text(mytree)`.

Consider predicting engagement from ratings and genre.

Split on genre by turning it into a group of numeric variables:

```
x <- model.matrix(PE ~ Genre + GRP, data=nbc)[,-1]
names(x) <- c("reality","comedy","GRP")
```

We have a reference factor level (`drama`)

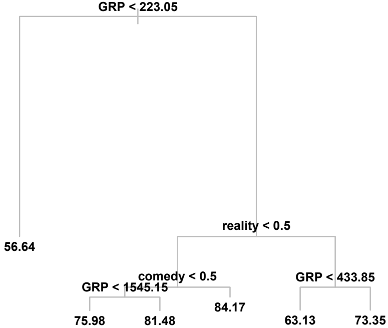
A regression tree:

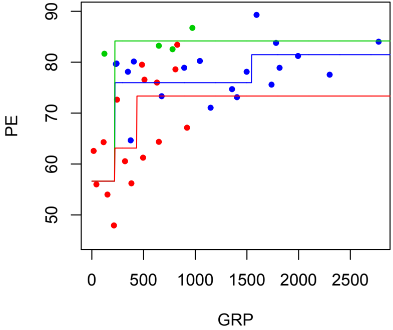
```
nbctree <- tree(PE ~ ., data=x, mincut=1)
```

Instead of genre, leaf predictions are expected engagement.

`mincut=1` allows for leaves containing a single show,
with expected engagement that single show's PE.

An NBC Show Engagement Tree





Green is comedy, blue is drama, red is reality

Nonlinear: PE increases with GRP, but in jumps

Follow how the tree translates into changing $\mathbb{E}[\text{PE}]$

Trees provide Automatic Interaction Detection

For example, different genres are more/less dependent on GRP.

AID was an original motivation for building decision trees.
Older algorithms have it in their name: CHAID, ...

This is pretty powerful technology: **nonlinearity** and **interaction** without having to specify it in advance.

Methods with these characteristics are called nonparametric.
No assumed parametric model (eg, $y = x\beta + \varepsilon,\ \varepsilon \sim \mathrm{N}(0, \sigma^2)$).

Pruning your tree for cross validation

Biggest challenge with such flexible models is avoiding overfit.
For CART, the usual solution is to rely on cross validation.

The basic constraints (`mincut, mindev`) lead to a full tree fit.
Prune this tree by removing split rules from the bottom up:

> At each step, remove the split that contributes least to deviance reduction, thus reversing CART's growth process.

Pruning yields candidate trees, and we use CV to choose.
Each prune step produces a candidate tree model, and we can compare their out-of-sample prediction performance.

Example: Prostate Cancer Prognosis

After tumor detection, there are many treatment options.

- Various chemo + radiation, surgical removal.

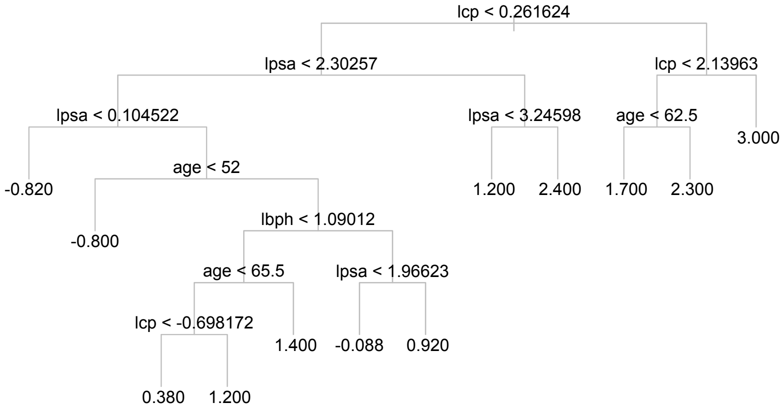
Biopsy information is available to help in deciding treatment

- Gleason Score: microscopic pattern classes.
- Prostate Specific Antigen: protein production.
- Capsular Penetration: reach of cancer into gland lining.
- Benign Prostatic Hyperplasia Amount: size of prostate.

Another influential variable is the patient's age.

The goal is to predict tumor log-volume (size, spread).

Full tree fit to 97 prostate cancer patients



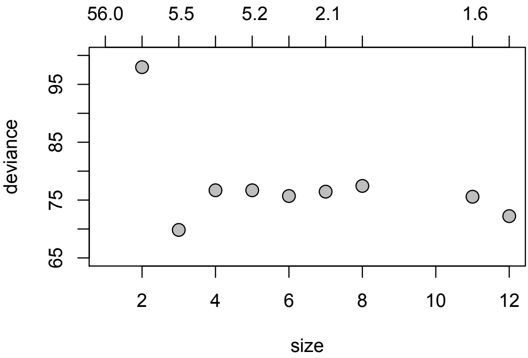
Leaf node labels are expected tumor `log(volume)`.

Do we need all the splits? Is the tree just fitting noise?

Cross-Validated Tree Pruning

cv.tree does cross-validation across pruning levels.

```
cvpst <- cv.tree(pstree, K=90) # K is nfolds
```



The output can be plotted, and it holds out-of-sample deviance for each tree size (the number of leaf nodes).

Pruning the Prostate Cancer Tree

Out-of-sample deviance can be used to choose tree size.

```
> cvpst$size
 [1] 12 11  8  7  6  5  4  3  2  1
> cvpst$dev
 [1] 72  75  77  76  76  77  77  70  97 160
```

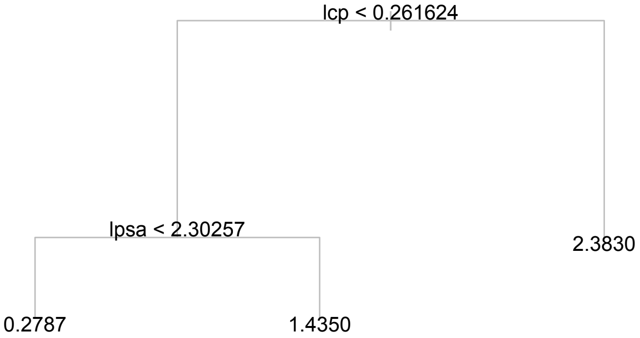
Since size 3 has lowest CV deviance, it is 'best'.

To fit this tree, use the `prune.tree` function:

```
pstcut <- prune.tree(pstree, best=3)
```

`pstcut` is then itself a new tree object.

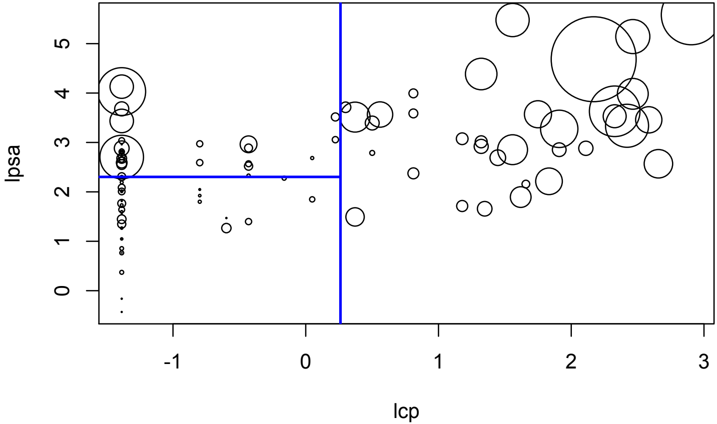
The Pruned Treatment Tree



CV chooses PSA and penetration as deciding variables.

Note the interaction: penetration effect depends on PSA.

Prostate Cancer Prognosis Tree



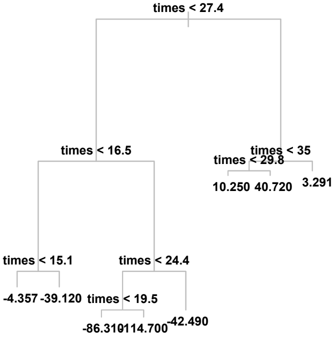
With only 2 relevant inputs, we can plot the data and tree fit.
Points proportional to tumor size, leaf partitions are in blue.

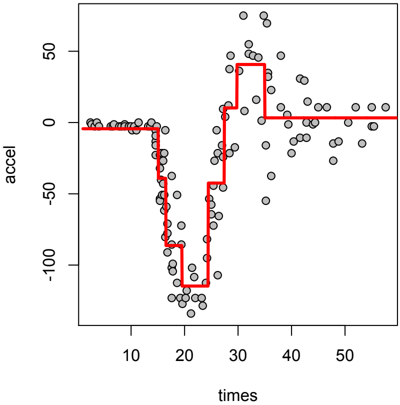
Trees are awesome

They automatically learn non-linear response functions and will discover interactions between variables.

Example: Motorcycle Crash Test Dummy Data

x is time from impact, *y* is acceleration on the helmet.





Unfortunately, it is tough to avoid overfit with CART:

Deep tree structure is so unstable that optimal depth is not easily chosen via cross validation, and there's no theory to fall back on.

Instead, we can average over a bootstrapped sample of trees:

- repeatedly re-sample the data, with-replacement,
  to get a 'jittered' dataset of n observations.
- for each resample, fit a CART tree.
- when you want to predict y for some $\mathbf{x}$,
  take the average prediction from this forest of trees.

Real structure that persists across datasets shows up in the average. Noisy useless signals will average out to have no effect.

This is a Random Forest

Random Forests

- Sample B subsets of the data + variables:
  e.g., observations $1, 5, 20, \ldots$

- Fit a tree to each subset, to get B fitted trees is $\mathcal{T}_b$. At each split, sample a subset of candidate variables for splitting

- Average prediction across trees:
  - for regression average $\mathbb{E}[y|\mathbf{x}] = \frac{1}{B}\sum_{b=1}^{B} \mathcal{T}_b(\mathbf{x})$.
  - for classification let $\{\mathcal{T}_b(\mathbf{x})\}_{b=1}^{B}$ vote on $\hat{y}$.

The observation resample is usually *with-replacement*, so that this is taking the *average of bootstrapped trees* (i.e., 'bagging')

Understanding Random Forests

Recall how CART is used in practice.

- Split to lower deviance until leaves hit minimum size.
- Create a set of candidate trees by pruning back from this.
- Choose the best among those trees by cross validation.

Random Forests avoid the need for CV.

Each tree '*b*' is not overly complicated because
you only work with a limited set of variables.

Your predictions are not 'optimized to noise' because
they are averages of trees fit to many different subsets.

RFs are a great go-to model for nonparametric prediction.

Model Averaging

This technique of 'Model Averaging' is central to many advanced nonparametric learning algorithms.

ensemble learning, mixture of experts, Bayesian averages, ...

It works best with flexible but simple models

Recall lasso as a stabilized version of stepwise regression (if you jitter the data your estimates stay pretty constant).

Model averaging is a way to take arbitrary *unstable* methods, and make them stable. This makes training easier.

Probability of rain on a new day is the average P(rain) across some trees that split on forecast, others on sky.

We don't get tied to one way of deciding about umbrellas.

Random Forests in R

R has the `randomForest` package,
which works essentially the same as `tree`

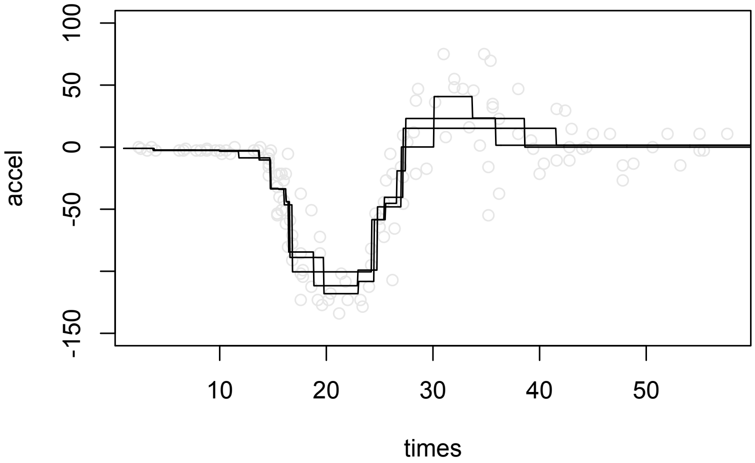
```
    rf <- randomForest(outcome ~ ., data=data)
```

For big datasets, use $\texttt{x} = \texttt{x},\ \texttt{y} = \texttt{y}$ like in `gamlr`.

Unfortunately, you lose the interpretability of a single tree.
However, if you set `importance=TRUE`, `Random Forest` will evaluate each $\mathcal{T}_b$'s performance on the *left-out sample* (recall each tree is fit on a sub-sample). This yields nice OOS stats.

They can be slow (due to many tree fits) but
they can also be fit in parallel or on distributed data...

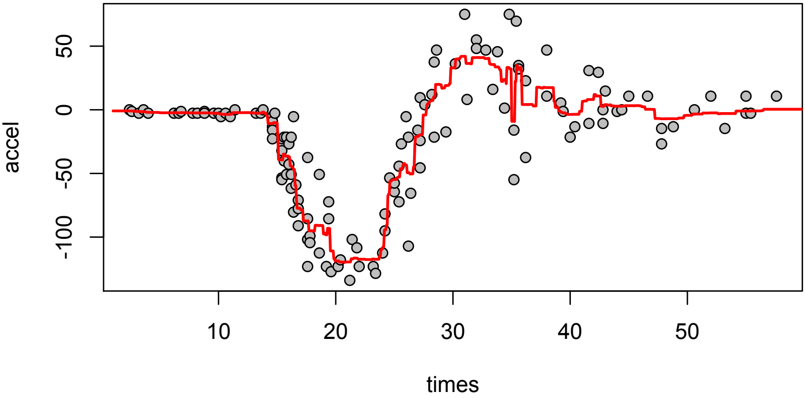
Random Trees for the Motorcycle Data



If you fit to random subsets of the data,

you get a slightly different tree each time.

Model Averaging with Random Forests



Averaging many trees yields a single response surface.

Still looks like a bit of overfit to me, which remains a danger.

A larger example: California Housing Data

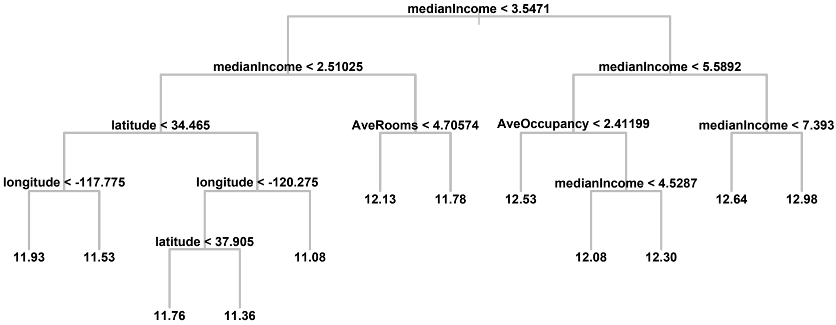
Median home values in census tracts, along with

- Latitude and Longitude of tract centers.
- Population totals and median income.
- Average room/bedroom numbers, home age.

The goal is to predict log(MedVal) for census tracts.

Difficult regression: Covariate effects change with location.
How they change is probably not linear.

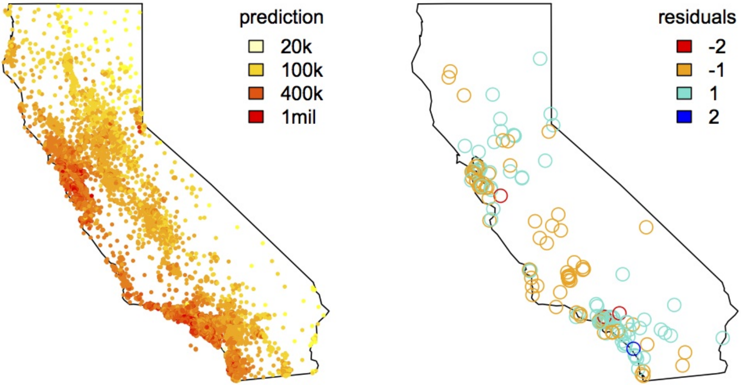
CART Dendrogram for CA housing



Income is dominant, with location important for low income.

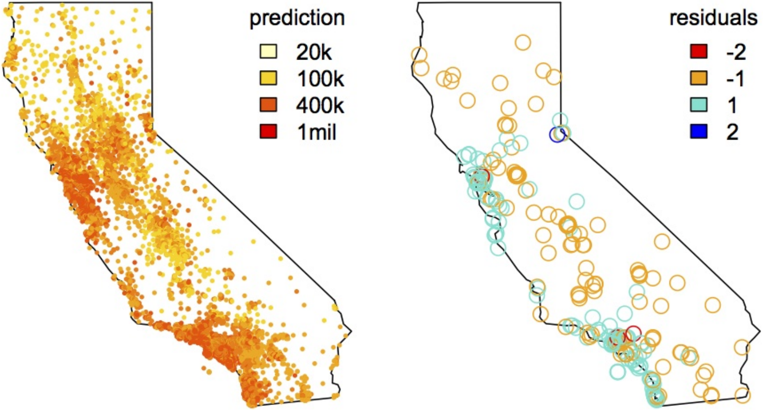
Cross Validation favors the most complicated tree: 12 leaves.

LASSO fit for CA housing data



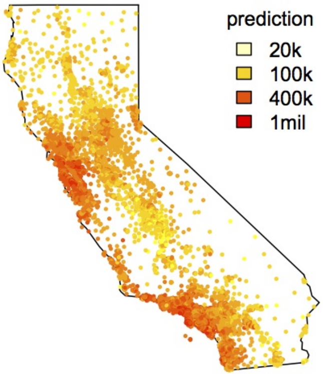
Looks like over-estimates in the Bay, under-estimates in OC.

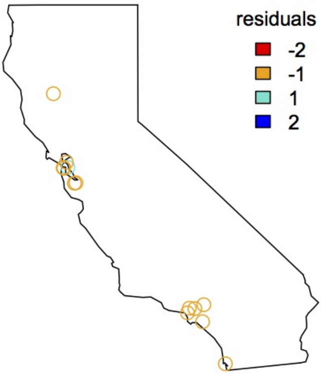
CART fit for CA housing data



Under-estimating the coast, over-estimating the central valley?

randomForest fit for CA housing data

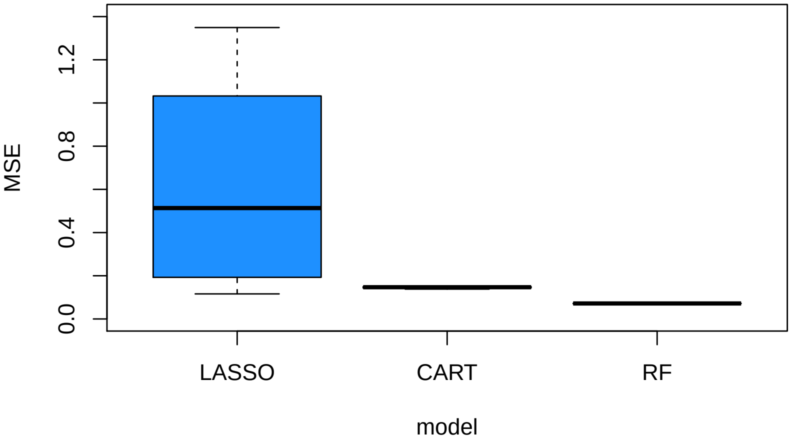




No big residuals! (although still missing the LA and SF effects)

Overfit? From out-of-sample prediction it appears not.

CA housing: out-of-sample prediction

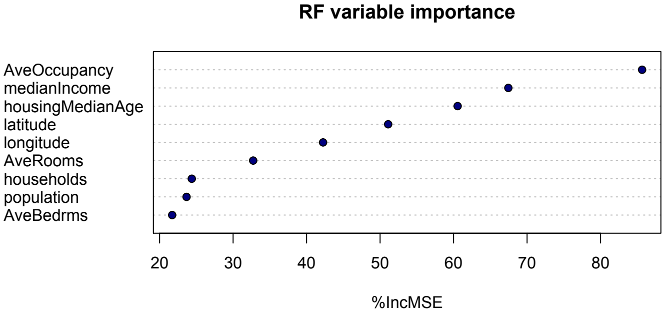


Trees outperform LASSO: gain from nonlinear interaction.

RF is better still than CART: benefits of model averaging.

Although you don't have a nice single tree to interpret, `randomForest` provides OOS variable importance plots.

You need to run randomForest with `importance=TRUE`. Otherwise it doesn't store the necessary information.



The x-axis here is the % amount that removing splits on that variable would increase the MSE.

For classification it plots increase in % misclassified.

Roundup on Tree-based learning

We've seen two techniques for building tree models.

- CART: recursive partitions, pruned back by CV.
- randomForest: average many simple CART trees.

There are many other tree-based algorithms.

- Boosted Trees: repeatedly fit simple trees to residuals.
  Fast, but it is tough to avoid over-fit (requires full CV).
- Bayes Additive Regression Trees: mix many simple trees.
  Robust prediction, but suffers with non-constant variance.
- Dynamic Trees: grow sequential 'particle' trees
  Good online, but fit depends on data ordering

Trees are poor in high dimension, but fitting them to low dimension factors (principle components) is a good option.

Roundup on Nonlinear Regression and Classification

Many other *nonparametric learning* algorithms

- Neural Networks (and deep learning):
  many recursive logistic regressions.
- Support Vector Machines:
  Project to HD, then classify.
- Gaussian Processes, splines, wavelets, etc:
  Use sums of curvy functions in regression.

Some of these are great, but all take a ton of tuning.

Nothing's better out-of-the-box in low dimension than trees. But: when the (simpler) linear model fits, it will do better.
This is most often the case in very high dimension.

Homework due next week

Use your project data for this homework!

If you don't have it yet, get it now.

Build and interpret both a single tree and a random forest.

You can compare the result to other techniques we've learned.

CART: fit, prune, + plot. Concentrate on interpretation.

RF: plot variable importance and predictive performance.