Lecture 6

DJM

30 October 2018

Non-linear classifiers

Non-linear classifiers

Many methods have linear decision boundaries

We know that sometimes this isn't sufficient to represent data

Sometimes we include a polynomial effect or a log transform in multiple regression to address such deficiencies

Sometimes, a linear boundary, but in a different space makes all the difference.

KNN

- A supremely simple non-linear classifier

- **Idea**: to predict at $x \mid \mathcal{D}$, average the k nearest points in $\mathcal{D}$

- This is very easy to code. There are sophisticated algorithms (not simple) that make this really fast.

- R package `class` implements this via `knn()`

- Main issue: curse of dimensionality. When there are many dimensions, things are not very "close" anymore.

KNN comparison
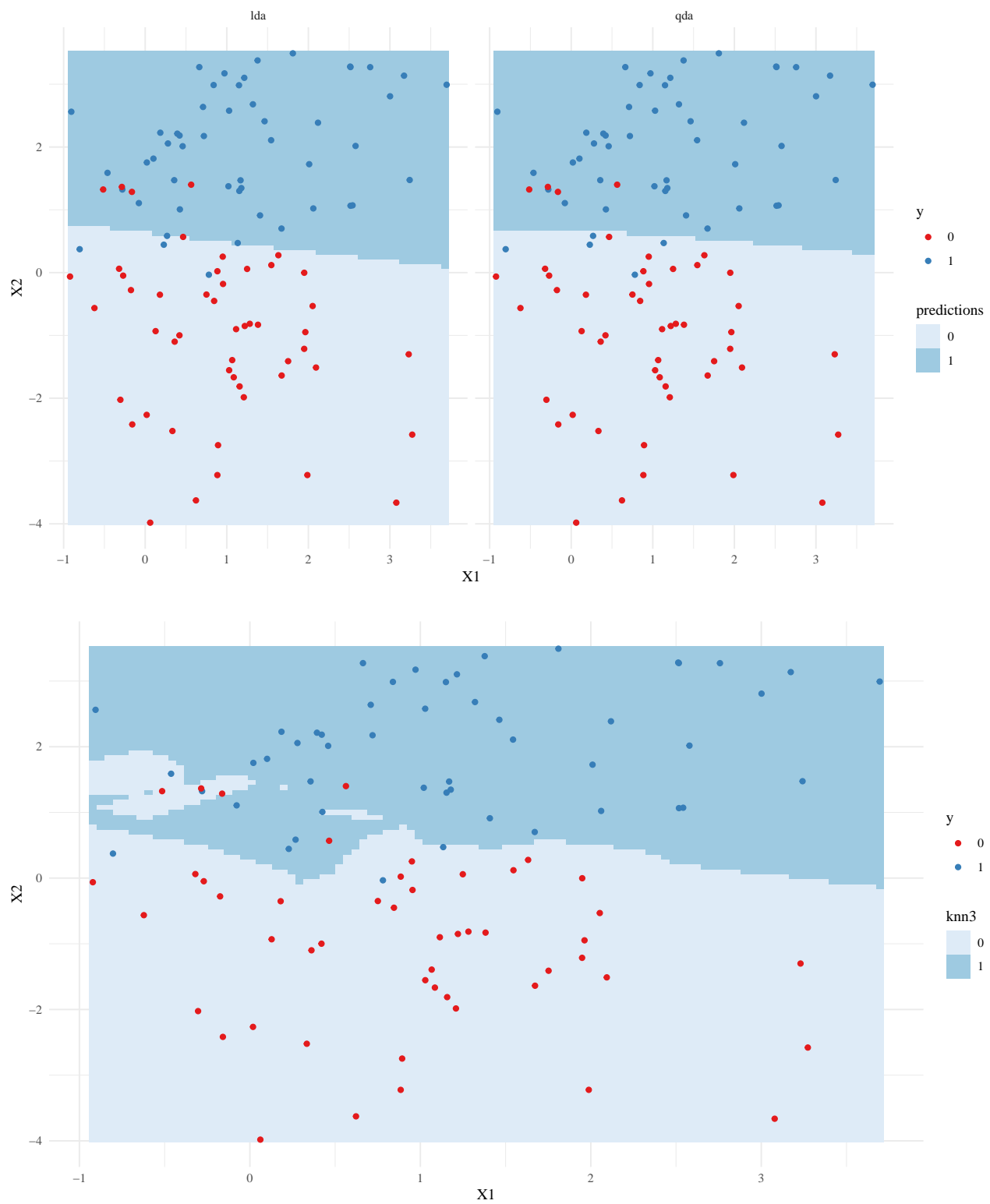
Choosing k

- Choosing k is very important.
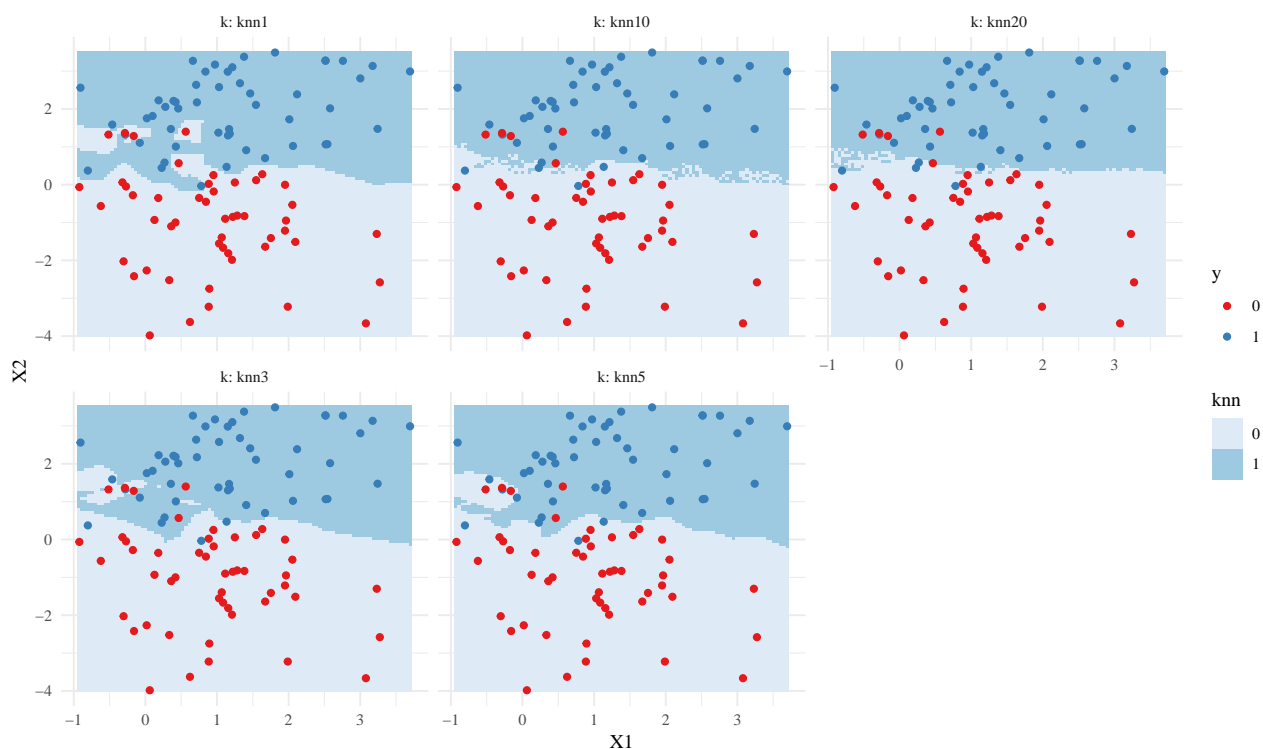
- How should we choose k?

- Scaling is also very important. The nearest neighbors are determined by their distance, so better to standardize your data first.

knn.cv

```
kmax = 20
err = double(kmax)
for(ii in 1:kmax){
  pk = knn.cv(df[,-1],df$y, k=ii) # does leave one out CV
  err[ii] = mean(pk != df$y)
}
ggplot(data.frame(k=1:kmax,error=err), aes(k,error)) + geom_point(color=red) +
  geom_line(color=red)
```
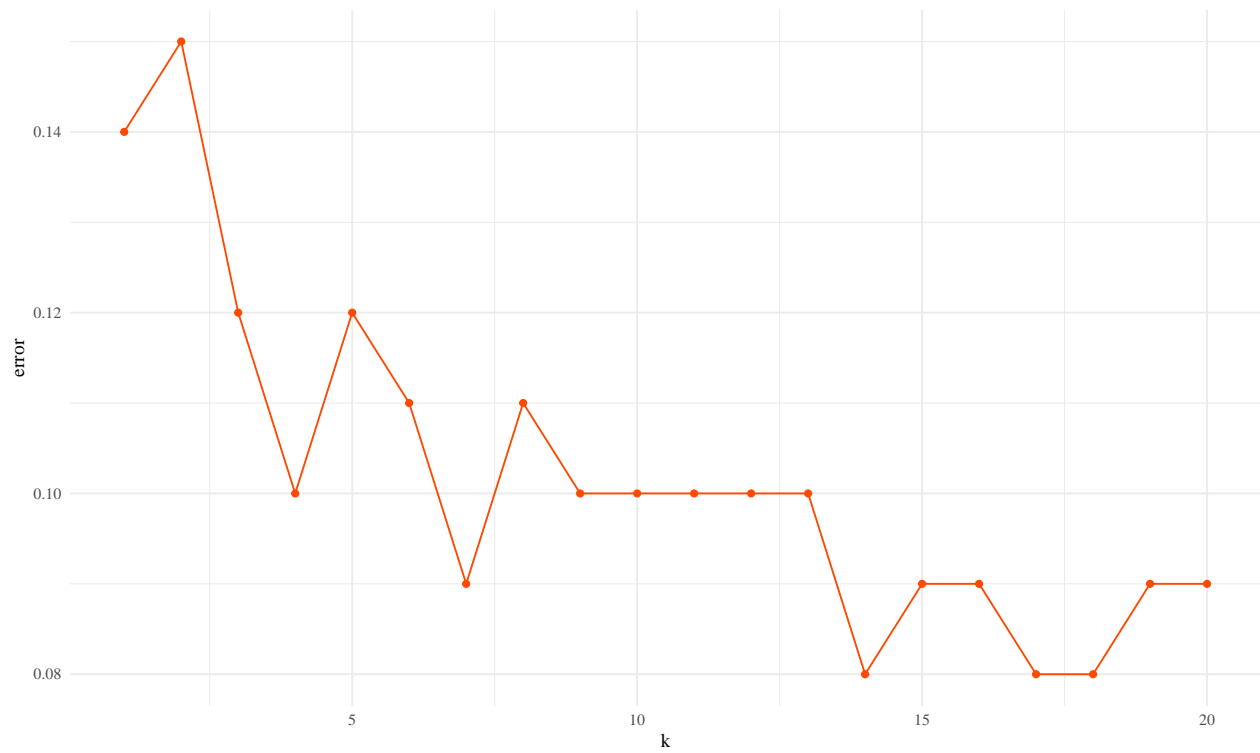
- I would use the ~~largest~~ k that is close to the minimum. This produces simpler, smoother, decision boundaries.
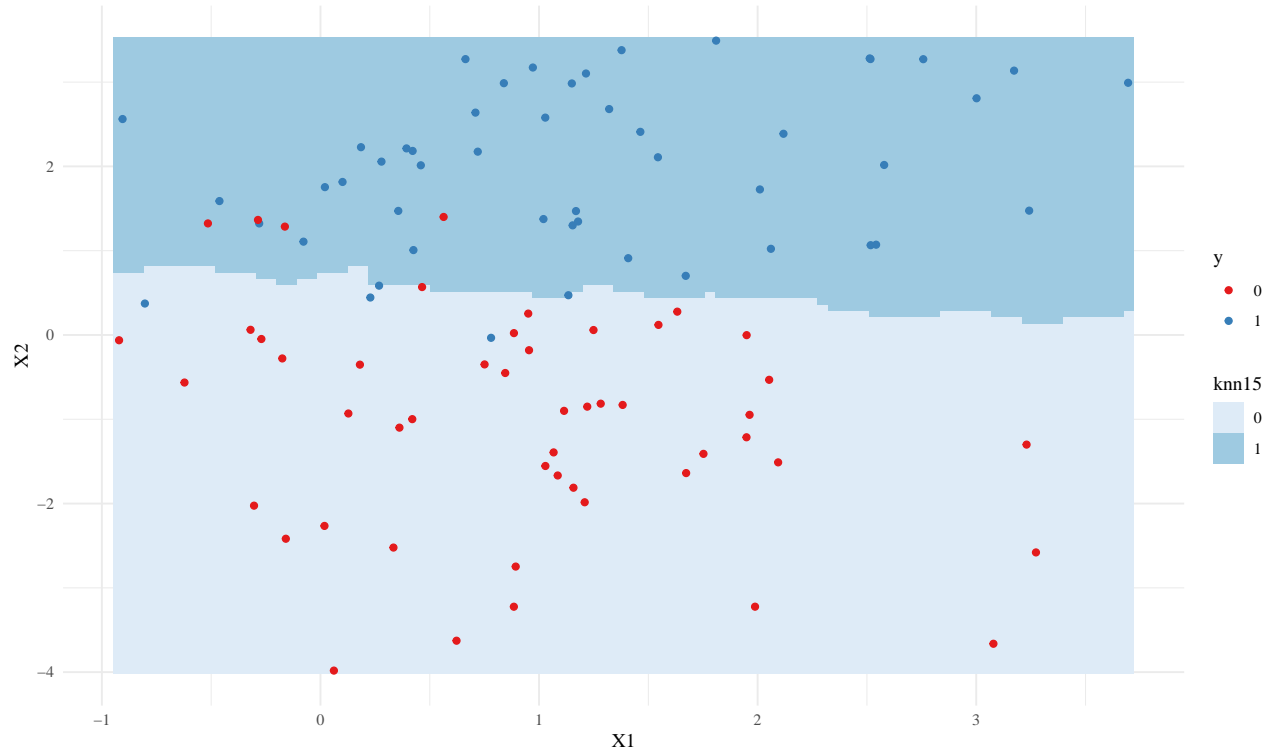

```
##          truth
## predicted  0  1
##          0 46  3
```

4

```
##        1  4 47
## [1] 0.07
```

Kernelization

Other non-linear classifiers

- We already saw that knn produces non-linear decision boundaries.
- "Kernelization" is a way of turning linear classifiers (or linear regression) into non-linear classifiers.
- You've already seen this happen when you add interactions or quadratic terms to linear models

The idea

- Suppose X_i takes values in some space $\mathcal{X}$.
- Find a mapping $\phi : \mathcal{X} \to \mathcal{Z}$.
- Apply a linear classifier on $\mathcal{Z}$.
- Example: $\mathcal{X} = \mathbb{R}$, $\mathcal{Z} = \mathbb{R}^2$, $\phi(x) = (z_1, z_2) = (x, x^2)$
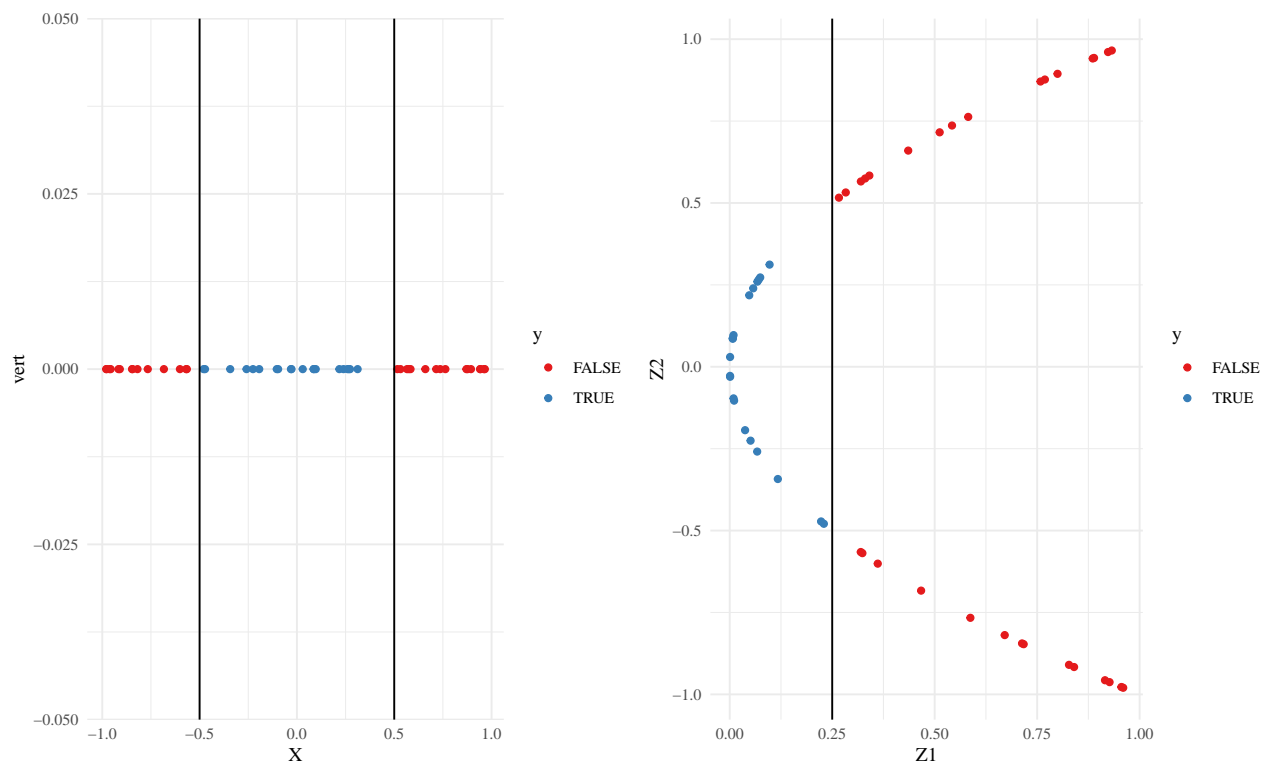
- A linear classifier in the higher-dimensional space corresponds to a non-linear classifier in low dimensions.
- Of course, if $\dim(\mathcal{Z})$ is too big, then there will be too many parameters to estimate well.

Transformations

Of course, as we include more transformations,

- We need to choose the transformations
- can become difficult if we aren't careful
- We need to to prevent overfitting

Can we do all this in an automated fashion?

The trick (high level)

If:

1. There is a mapping $\phi : \mathcal{X} \to \mathcal{Z}$
2. Your classifiers of choice only needs the ~~inner product~~ between observations, not the observations themselves. (logistic regression, LDA, and SVMs work)
3. There is a function K such that $K(x, x') = \langle \phi(x), \phi(x') \rangle$ (Mercer's theorem gives this)

Then, we can just replace all inner products $\langle x, x' \rangle$ with $K(x, x')$.

- This produces a nonlinear classifier based on a linear classifier.
- We don't actually need x or ϕ, just $K(x, x')$ (this is an $n \times n$ matrix)
- $\mathcal{Z}$ can be infinite dimensional.

SVM classification plot
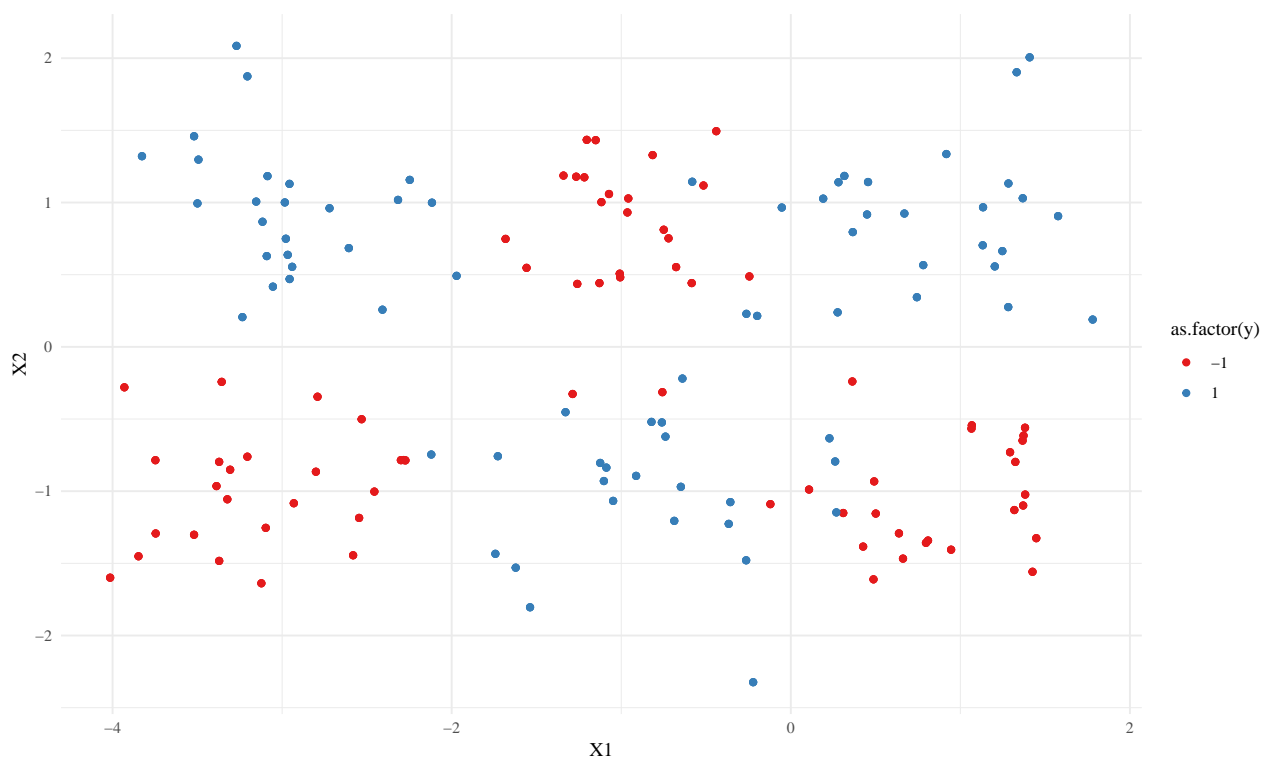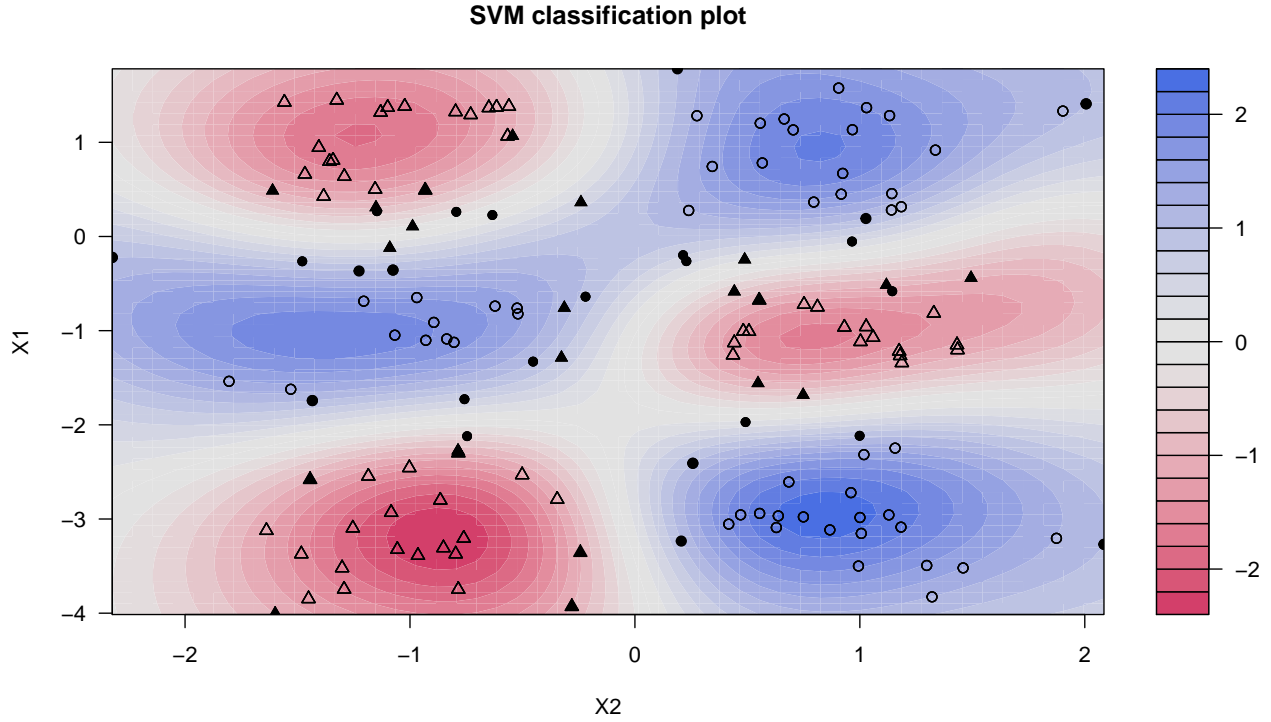
Preliminaries

Let $A \in \mathbb{R}^{p \times p}$ be a symmetric, nonnegative definite matrix:

$$z^\top A z \geq 0 \text{ for all } z \ \text{ and } \ A^\top = A$$

Then, A has an eigenvalue expansion

$$A = U D U^\top = \sum_{j=1}^{p} d_j u_j u_j^\top$$

where $d_j \geq 0$

Each such A, generates a new inner product:

$$\langle z, z' \rangle = z^\top z' = z^\top I z'$$

$$\langle z, z' \rangle_A = z^\top A z'$$

If we enforce A to be positive definite, then $\langle z, z \rangle_A = \|z\|_A^2$ is a norm.

Kernel function

We can generalize the notion of p.s.d. matrices to functions.

We call $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a (psd) kernel function if

$$\sum_{i}^{n} \sum_{j}^{n} a_i a_j k(x_i, x_j) \geq 0, \quad \forall a \in \mathbb{R}^n, \ x_1, \dots, x_n \in \mathcal{X}.$$

7

Just as we can ask for the eigendecomposition of a p.s.d. matrix, we can do the same for these kernel functions.

Write the eigenvalue expansion of k as

$$k(X, X') = \sum_{j=1}^{\infty} \theta_j \phi_j(X) \phi_j(X')$$

with

- $\theta_j \geq 0$ (nonnegative definite)
- $\left\| (\theta_j)_{j=1}^{\infty} \right\|_2 = \sum_{j=1}^{\infty} \theta_j^2 < \infty$
- The ϕ_j are orthogonal : $\int \phi_j \phi_{j'} = \delta_{j,j'}$

Mercer's theorem states that any (continuous) psd Kernel has an eigenexpansion in this sense.

And such a k is called a Mercer kernel

Properties

$$k(X, X') = \sum_{j=1}^{\infty} \theta_j \phi_j(X) \phi_j(X')$$

- There is an equivalence between k and ϕ
- If you given me appropriate ϕ functions and θ, then I can (in principle) recover k and vice versa.
- Seems totally esoteric for the moment.

Dumb example

- $\theta_j = I(j \leq n)$
- $\phi_j(x) = x_j I(j \leq n)$

$$k(X, X') = \sum_{j=1}^{\infty} \theta_j \phi_j(X) \phi_j(X') = X^\top X'$$

This is the **linear** kernel.

More interesting (squared terms and interactions)

- Suppose you want to do linear regression with two predictors, but you think there may be nonlinearities.

- So you create more features by including x_1^2, x_2^2, and $x_1 x_2$. You are now doing regression in $p = 5 + 1$ dimensions instead of $p = 2 + 1$ (including the intercept).

- (Trick) Define
$$\Phi(x) = \Phi((x_1, x_2)) = (1, \ \sqrt{2}x_1, \ \sqrt{2}x_2, \ x_1^2, \ x_2^2, \ \sqrt{2}x_1 x_2)^\top$$

- Then
$$\begin{aligned} k(u, v) &= \langle \Phi(u), \ \Phi(v) \rangle \\ &= \Phi(u)^\top \Phi(v) \\ &= \sum_{k=1}^{M} \Phi_k(u) \Phi_k(v) \\ &= 1 + 2u_1 v_1 + 2u_2 v_2 + u_1^2 v_1^2 + u_2^2 v_2^2 + 2u_1 u_2 v_1 v_2 \end{aligned}$$

- This is a special case ($d = p = 2 \Rightarrow M = 6$) of the **polynomial** kernel

$$k_d(X, X') = (X^\top X' + 1)^d,$$

which has $M = \binom{d+p}{d}$ eigenfunctions.

Simplification

- Kernels work by operating on the **rows** of the predictors in your regression.
- Thus, $k(X, X')$ is a function on two rows.
- Thus $\mathbb{K} = [k(X_i, X_j)]_{i,j}$ is always an $n \times n$ matrix
- It doesn't matter how many columns you have.
- It doesn't matter how many eigenfunctions you have (can be infinite)
- You can get these kernels either by specifying a function k that is p.s.d., or by specifying the eigenfunctions. You don't need to go back and forth.
- What good does $\mathbb{K}$ do?

(kernel) Ridge regression

- We showed a few weeks back that

$$\widehat{\beta}_r = \arg\min_\beta \frac{1}{2} \|Y - X\beta\|_2^2 + \frac{\lambda}{2} \|\beta\|_2^2 = (X^\top X + \lambda I)^{-1} X^\top Y$$

- Therefore to predict at some x_0

$$\widehat{f}_r(x_0) = x_0^\top (X^\top X + \lambda I)^{-1} X^\top Y.$$

- Applying the Woodbury Identity,

$$\widehat{f}_r(x_0) = x_0^\top (X^\top X + \lambda I)^{-1} X^\top Y = \frac{1}{\lambda} x_0^\top X^\top (\frac{1}{\lambda} X X^\top + I)^{-1} Y$$

- Thus ridge regression depends only on $XX^\top$ (and $x_0^\top X^\top$).
- This is exactly the linear kernel
- Everywhere we see $XX^\top$, we can plug in $\mathbb{K}$ for any kernel k we choose.
- Doesn't work for OLS (function of inner products of columns, not rows)

Kernels and the dual

- Most often, the representation of the problem that allows for kernelization is the dual of the optimization problem

Dual of ridge regression

1. Convert to constrained problem

$$\min_\beta \frac{1}{2} \|y - X\beta\|_2^2 + \frac{\lambda}{2} \|\beta\|_2^2 \iff \min_{r,\beta} \frac{1}{2} \|r\|_2^2 + \frac{\lambda}{2} \|\beta\|_2^2 \quad \text{s.t.} \quad r = y - X\beta$$

2. Form Lagrangian

$$L(r, \beta, \alpha) = \frac{1}{2} \|r\|_2^2 + \frac{\lambda}{2} \|\beta\|_2^2 + \alpha^\top (r - y + X\beta).$$

3. Eliminate r and β (take derivatives, set to zero, substitute back in)

$$\beta = \frac{1}{\lambda} X^\top \alpha, \qquad r = -\alpha,$$

$$\min_\alpha \frac{1}{2} \alpha^\top (\mathbb{K} + \lambda I)^{-1} \alpha - \lambda \alpha^\top y$$

where $\mathbb{K} = XX^\top$.

4. Solve for α and recover β

$$\alpha = \lambda(\mathbb{K} + \lambda I)^{-1} y, \qquad \beta = X^\top (\mathbb{K} + \lambda I)^{-1} y.$$

Kernel SVM

Support vector machines minimize (in Lagrangian form, α_i Lagrange multipliers)

$$\frac{1}{2} \|\beta\|_2^2 - \sum_{i=1}^{n} \alpha_i [Y_i (X_i^\top \beta + \beta_0) - 1]$$

Derivatives with respect to β and β_0 imply:

- $\beta = \sum_{i=1}^{n} \alpha_i Y_i X_i$
- $0 = \sum_{i=1}^{n} \alpha_i Y_i$

Substituting back into the Lagrangian:

$$\text{wolfe dual} = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{k=1}^{n} \alpha_i \alpha_k Y_i Y_k X_i^\top X_k$$

(this is all subject to $\alpha_i \geq 0$)

SVMs maximize the wolfe dual

Note that it depends only on $X_i^\top X_j$

Write the solution function

$$h(X) = \beta_0 + \beta^\top X = \beta_0 + \sum_{i=1}^{n} \alpha_i Y_i X_i^\top X$$

the support vector classifier $\Rightarrow$:

$$h(X) = \beta_0 + \sum_{i=1}^{n} \alpha_i Y_i k(X_i, X)$$

Kernel SVMs

- We need only define k that is symmetric, positive definite

Some common choices for SVMs:

- $k(u, v) = (1 + u^\top v)^d$ (we saw this has a finite eigen expansion)
- $k(u, v) = e^{-\tau \|x - y\|_b^b}$ (this one is infinite, the basis functions are the Fourier series)

For example, $b = 2$ and $\tau = 1/(2\sigma^2)$ is (proportional to) the Gaussian density

Kernel SVMs: Summary

The solution form for SVM is

$$\beta = \sum_{i=1}^{n} \alpha_i Y_i X_i$$

Kernelized, this is

$$\beta = \sum_{i=1}^{n} \alpha_i Y_i \Phi(X_i)$$

Therefore, the induced hyperplane is:

$$h(X) = \Phi(X)^\top \beta + \beta_0 = \sum_{i=1}^{n} \alpha_i Y_i \langle \Phi(X), \Phi(X_i) \rangle + \beta_0$$

$$= \sum_{i=1}^{n} \alpha_i Y_i k(X, X_i) + \beta_0$$

The final classification is $\widehat{g}(X) = \mathrm{sgn}(\widehat{h}(X))$

SVMs in practice

The basic SVM functions are in the C++ library `libsvm`

The `R` package `e1071` calls those functions

An alternative is `kernlab`, which is what I'll demonstrate with

There is also a path algorithm in `svmpath`

For a nice comparison of these approaches, see "Support vector machines in `R`"

A few kernel SVMs

```
svp = ksvm(y~.,data=df,type="C-svc", kernel='vanilladot') # linear kernel

##  Setting default kernel parameters
plot(svp, data=df)
```

SVM classification plot
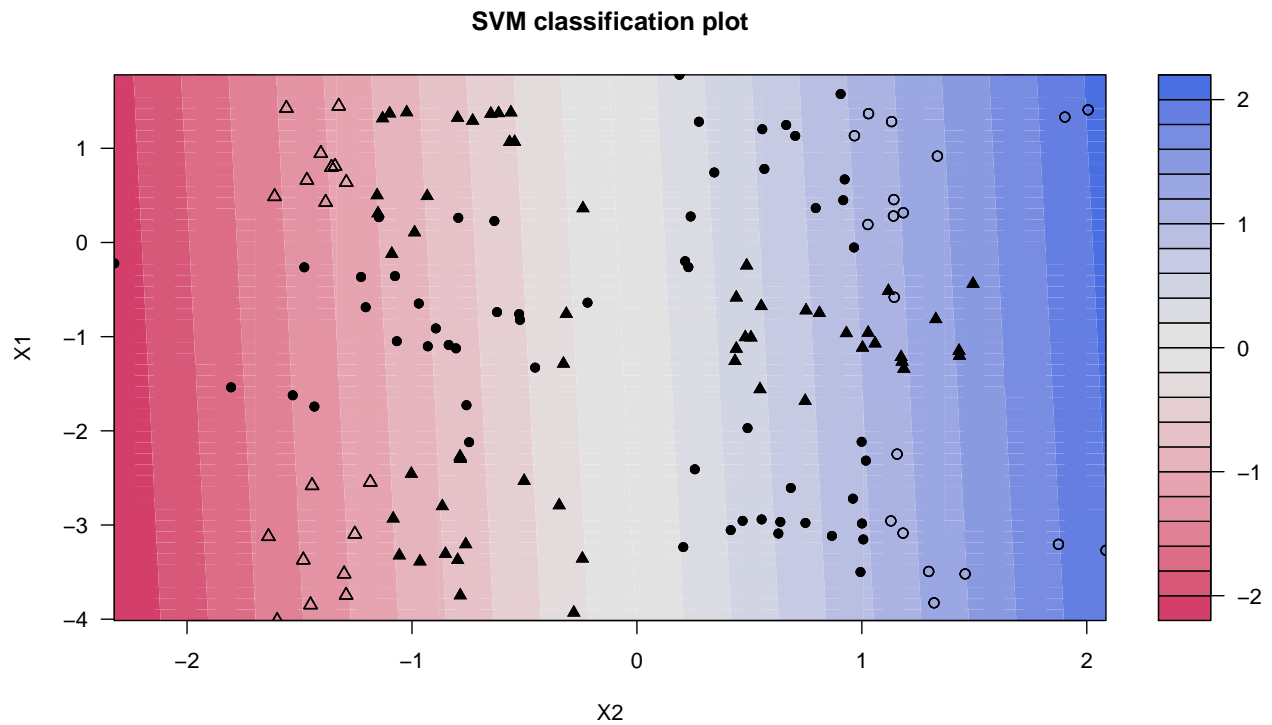
A few kernel SVMs

- It computes the tuning parameter(s) based on some heuristics.

```
svp = ksvm(y~.,data=df,type="C-svc") # Default is 'rbfdot'
plot(svp,data=df)
```

SVM classification plot
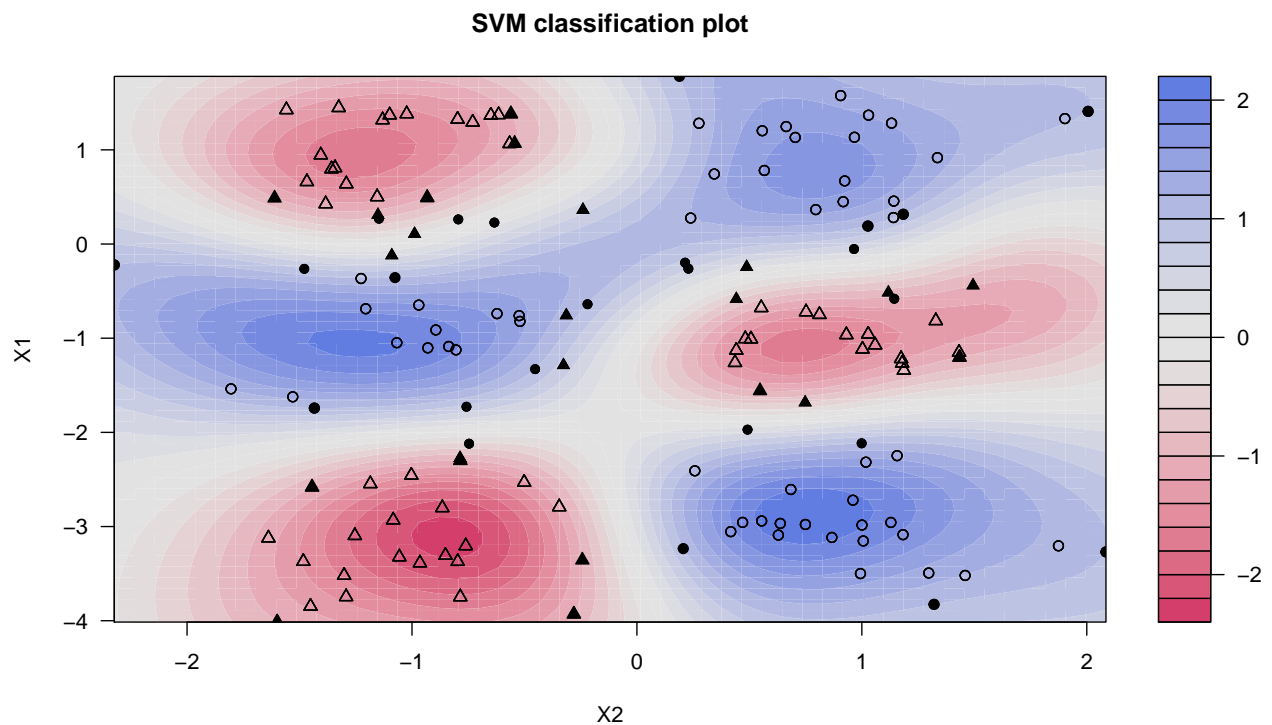
A few kernel SVMs

```
svp = ksvm(y~.,data=df,type="C-svc", kernel='polydot',
            kpar=list(degree=2)) # 2nd degree polynomial
plot(svp,data=df)
```

SVM classification plot
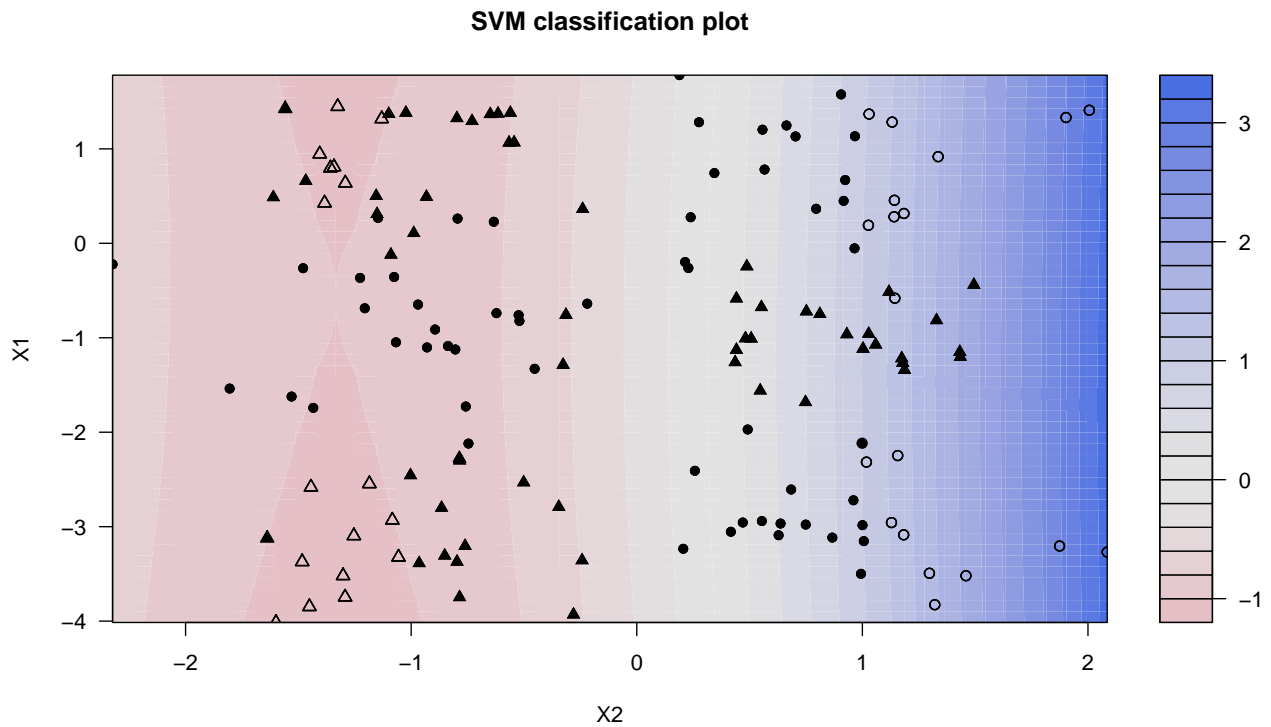
Trees

What is a (decision) tree?

- Trees involve stratifying or segmenting the predictor space into a number of simple regions.
- Trees are simple and useful for interpretation.

- Basic trees are not great at prediction.
- More modern methods that use trees are much better.

Example tree

Decision Tree: The Obama-Clinton Divide

In the nominating contests so far, Senator Barack Obama has won the vast majority of counties with large black or highly educated populations. Senator Hillary Rodham Clinton has a commanding lead in less-educated counties dominated by whites. Follow the arrows for a more detailed split.
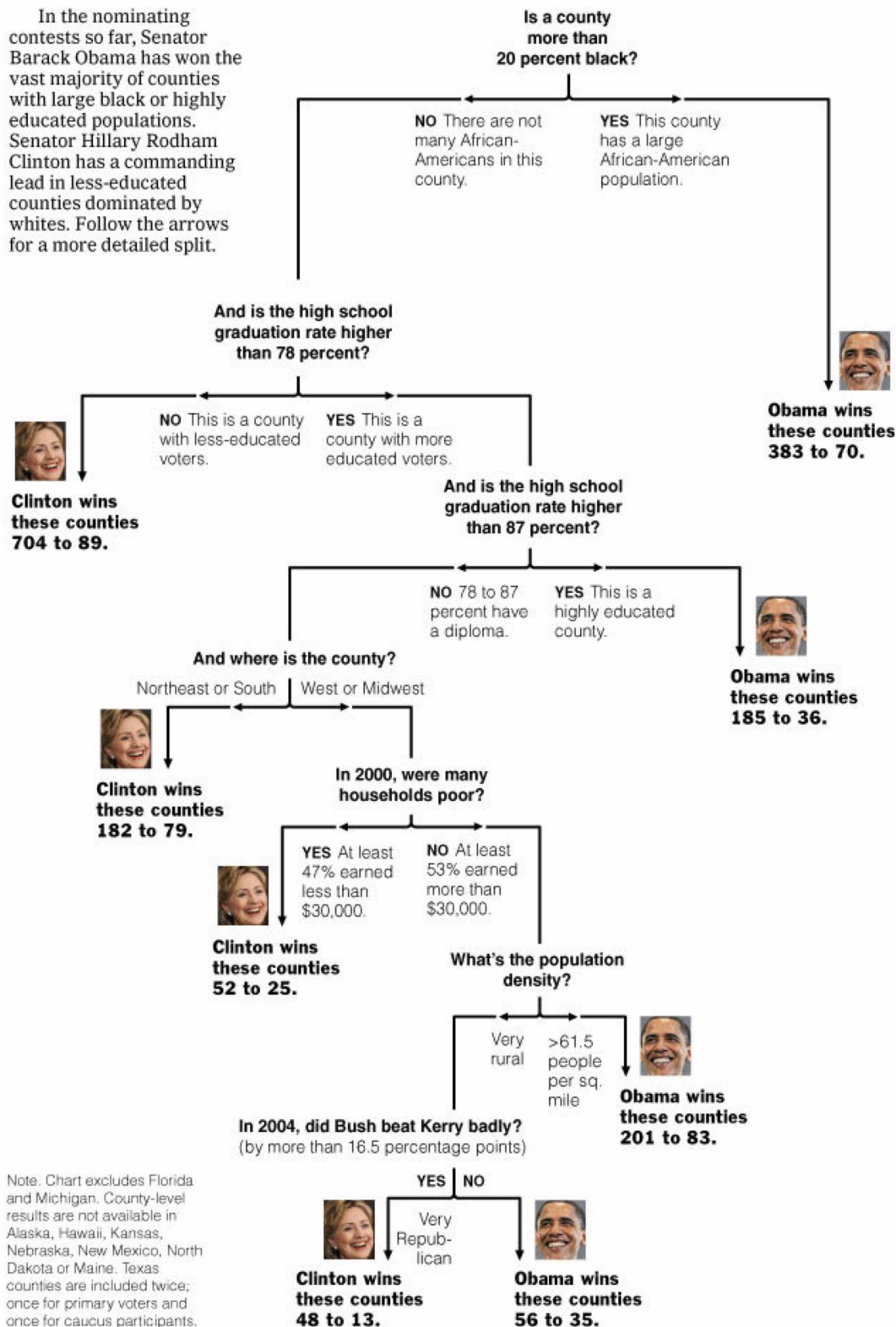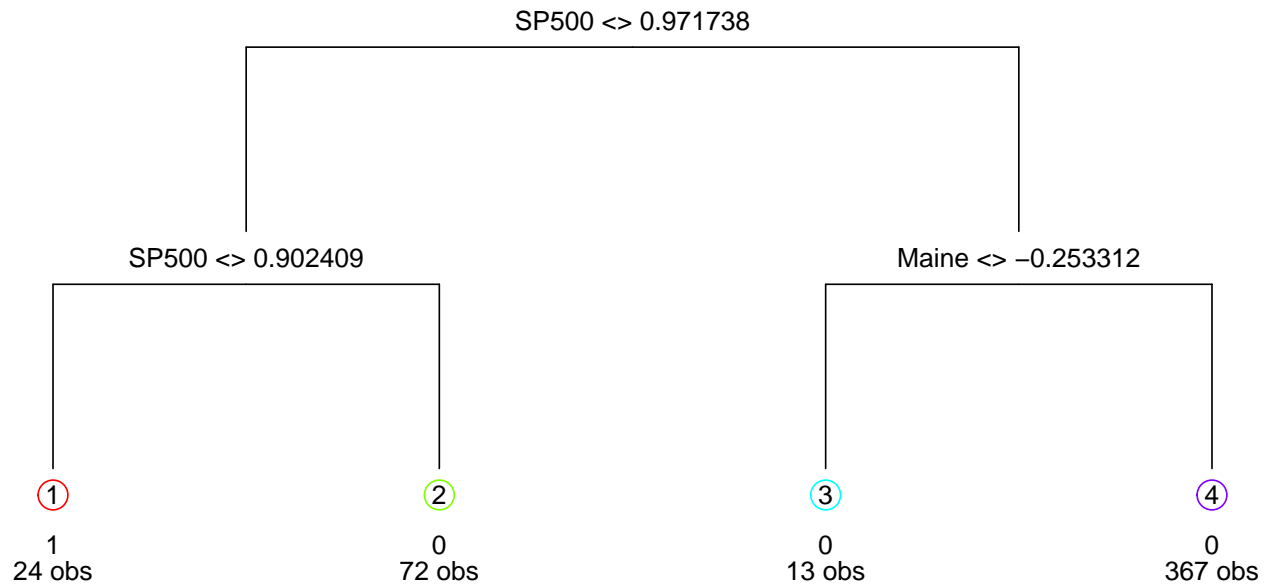
Is a county more than 20 percent black?

NO There are not many African-Americans in this county.

YES This county has a large African-American population.

Obama wins these counties 383 to 70.

And is the high school graduation rate higher than 78 percent?

NO This is a county with less-educated voters.

YES This is a county with more educated voters.

Clinton wins these counties 704 to 89.

And is the high school graduation rate higher than 87 percent?

NO 78 to 87 percent have a diploma.

YES This is a highly educated county.

Obama wins these counties 185 to 36.

And where is the county?

Northeast or South | West or Midwest

Clinton wins these counties 182 to 79.

In 2000, were many households poor?

YES At least 47% earned less than $30,000.

NO At least 53% earned more than $30,000.

Clinton wins these counties 52 to 25.

What's the population density?

Very rural | >61.5 people per sq. mile

Obama wins these counties 201 to 83.

In 2004, did Bush beat Kerry badly? (by more than 16.5 percentage points)

YES | **NO**

Very Republican

Clinton wins these counties 48 to 13.

Obama wins these counties 56 to 35.

Note. Chart excludes Florida and Michigan. County-level results are not available in Alaska, Hawaii, Kansas, Nebraska, New Mexico, North Dakota or Maine. Texas counties are included twice; once for primary voters and once for caucus participants.

Sources: Election results via The Associated Press; Census Bureau; Dave Leip's Atlas of U.S. Presidential Elections

AMANDA COX/
THE NEW YORK TIMES

Dendogram view

```
library(tree)
library(maptree)
small = recessions %>% filter(DATE<'2000-01-01') %>% select(USREC, Maine, SP500)
out.tree.plt = tree(USREC~.,data=small)
  # I picked these to make an interesting small tree
tmp.tree = prune.tree(out.tree.plt,best=4)
draw.tree(tmp.tree)
```

SP500 <> 0.971738

SP500 <> 0.902409   Maine <> −0.253312

①   ②   ③   ④

1   0   0   0
24 obs   72 obs   13 obs   367 obs

~~Terminology~~

- We call each split or end point a node. Each terminal node is referred to as a leaf.
  - This tree has 2 interior nodes and 4 terminal nodes.
- The interior nodes lead to branches.
  - This graph has two main branches (the first S&P 500 split).

Partitioning view

```
small$yhat = predict(tmp.tree, type='class')
plot(small$SP500, small$Maine, col=c(red,blue)[small$USREC],
     pch=19,cex=.6,ylab='Maine',xlab='sp500',
     bty='n',las=1)
partition.tree(tmp.tree,add=TRUE,col=green,cex=2)
```
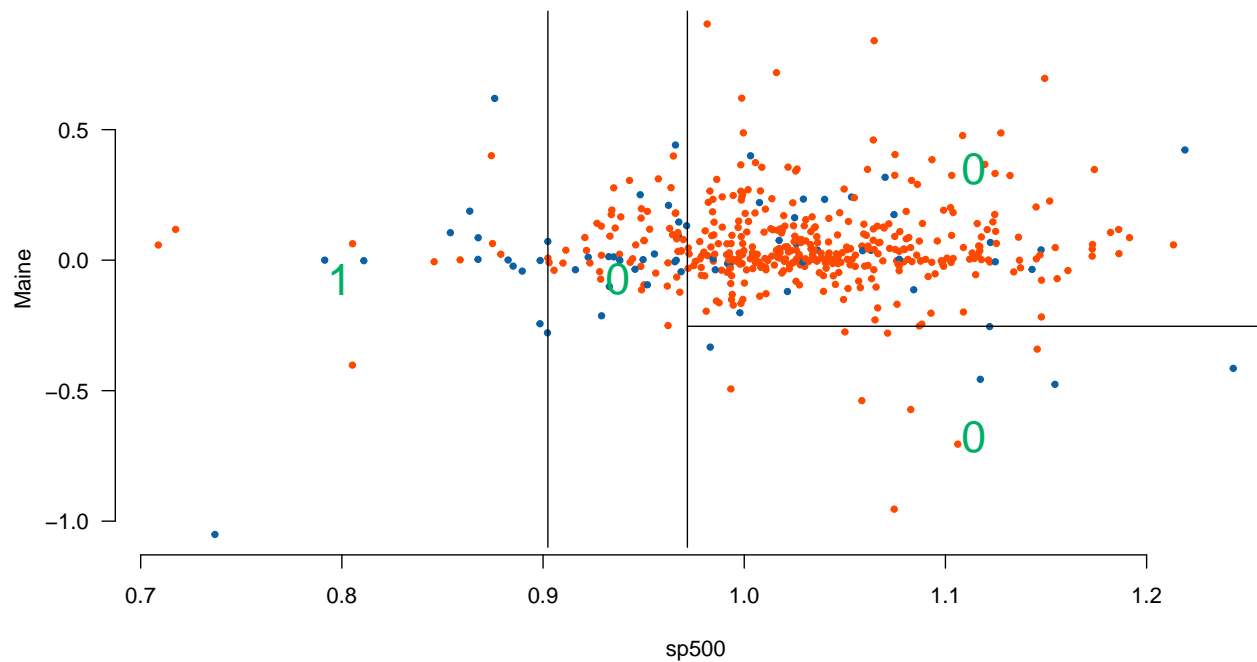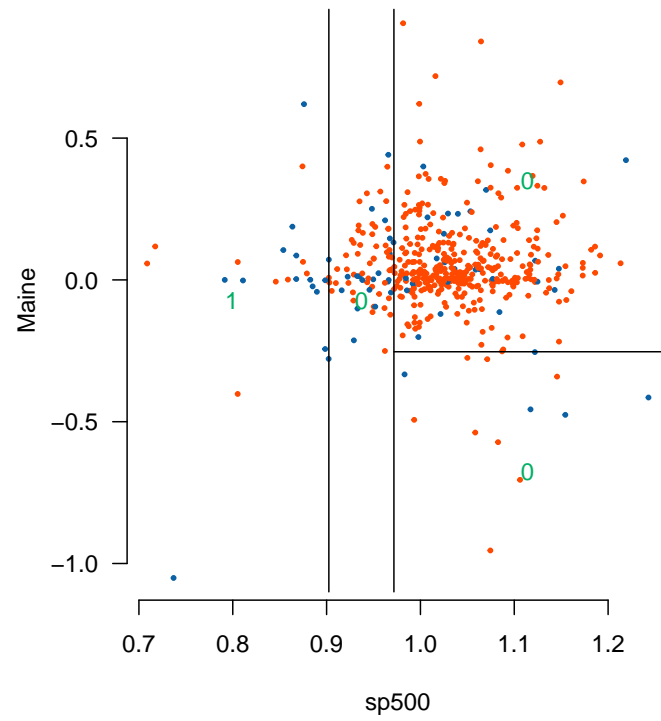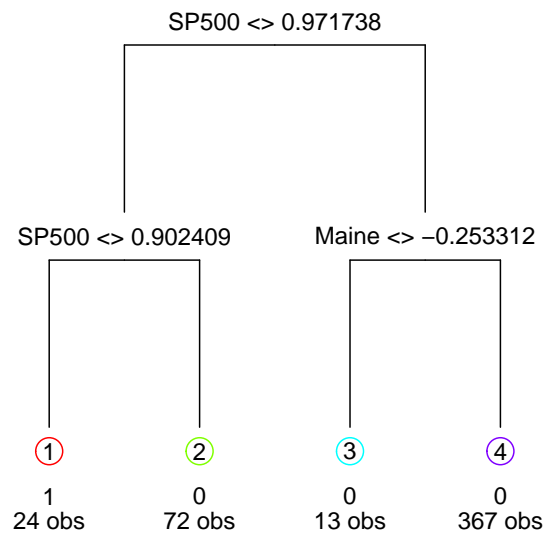
- We classify all observations in a region the same.
- The three regions are the leaves of the tree.
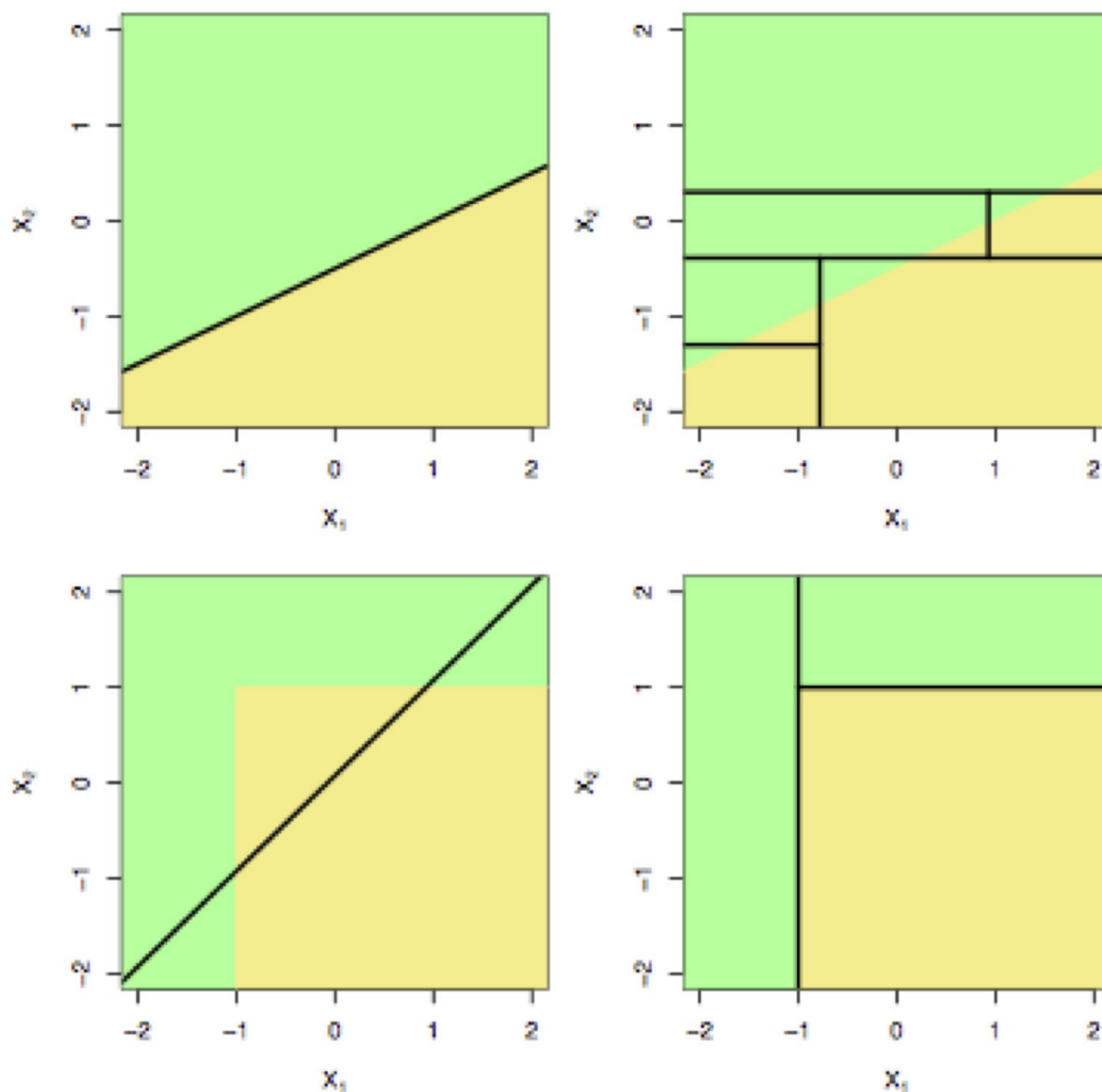
Tree

```
par(mfrow=c(1,2))
draw.tree(tmp.tree)
plot(small$SP500,small$Maine, col=c(red,blue)[small$USREC],
     pch=19,cex=.4,ylab='Maine',xlab='sp500',
     bty='n',las=1)
partition.tree(tmp.tree,add=TRUE,col=green)
```

SP500 <> 0.971738

SP500 <> 0.902409   Maine <> −0.253312

①   ②   ③   ④
1   0   0   0
24 obs   72 obs   13 obs   367 obs

We can interpret this as?

When do trees do well?

Top Row: A two-dimensional classification example in which the true decision boundary is linear. A linear boundary will outperform a decision tree that performs splits parallel to the axes.

Bottom Row: Here the true decision boundary is non-linear. A linear model is unable to capture the true decision boundary, whereas a decision tree is successful.

How do we build a tree?

1. Divide the predictor space into J non-overlapping regions $R_1, \ldots, R_J$ (this is done via greedy, recursive binary splitting).

2. Every observation that falls into a given region R_j is given the same prediction, which is determined by majority (or plurality) vote in that region.

Important:

- Trees can only make rectangular regions that are aligned with the coordinate axis.
- The fit is ~~greedy~~, which means that after a split is made, all further decisions are conditional on that split.

Does the tree fit?

How do we measure quality of fit?

There are many choices for a metric. Let p_{mk} be the proportion of training observations in the mth region that are from the kth class.

~~classification error rate:~~	$E = 1 - \max_k(\widehat{p}_{mk})$
~~Gini index:~~	$G = \sum_k \widehat{p}_{mk}(1 - \widehat{p}_{mk})$
~~cross-entropy:~~	$D = -\sum_k \widehat{p}_{mk} \log(\widehat{p}_{mk})$

Both Gini and cross-entropy can be thought of as measuring the purity of the classifier (small if all p_{mk} are near zero or 1). These are preferred over the classification error rate.

We build a classifier by growing a tree that minimizes G or D.

There's a problem

Following this procedure overfits!

- The process described so far will fit overly complex trees, leading to poor predictive performance.
- Overfit trees mean they have too many leaves.
- To stretch the analogy further, trees with too many leaves must be pruned.

Pruning the tree

- Cross-validation can be used to directly prune the tree, but it is far too expensive (computationally) to use in practice (combinatorial complexity).

- Instead, we use 'weakest link pruning',

$$\sum_{m=1}^{|T|} \sum_{i \in R_m} I(y_i = \widehat{y}_{R_m}) + \alpha|T|$$

where $|T|$ is the number of terminal nodes.

- Essentially, we are trading training fit (first term) with model complexity (second) term (compare to lasso).

- Now, cross-validation can be used to pick α.

Results of trees on recession data

```
big.tree = tree(USREC~.-DATE,data=recessions,subset=train)
tree.cv = cv.tree(big.tree, K=5)
best.k = tree.cv$k[which.min(tree.cv$dev)]
tree = prune.tree(big.tree, k=best.k)
predclass = data.frame(truth = recessions$USREC[test])
predclass$tree = predict(tree, newdata = recessions[test,], type='class') ## for later
```

```
par(mfrow=c(1,2),mar=c(0,0,0,0))
draw.tree(big.tree,cex=.4)
draw.tree(tree,cex=.5)
```
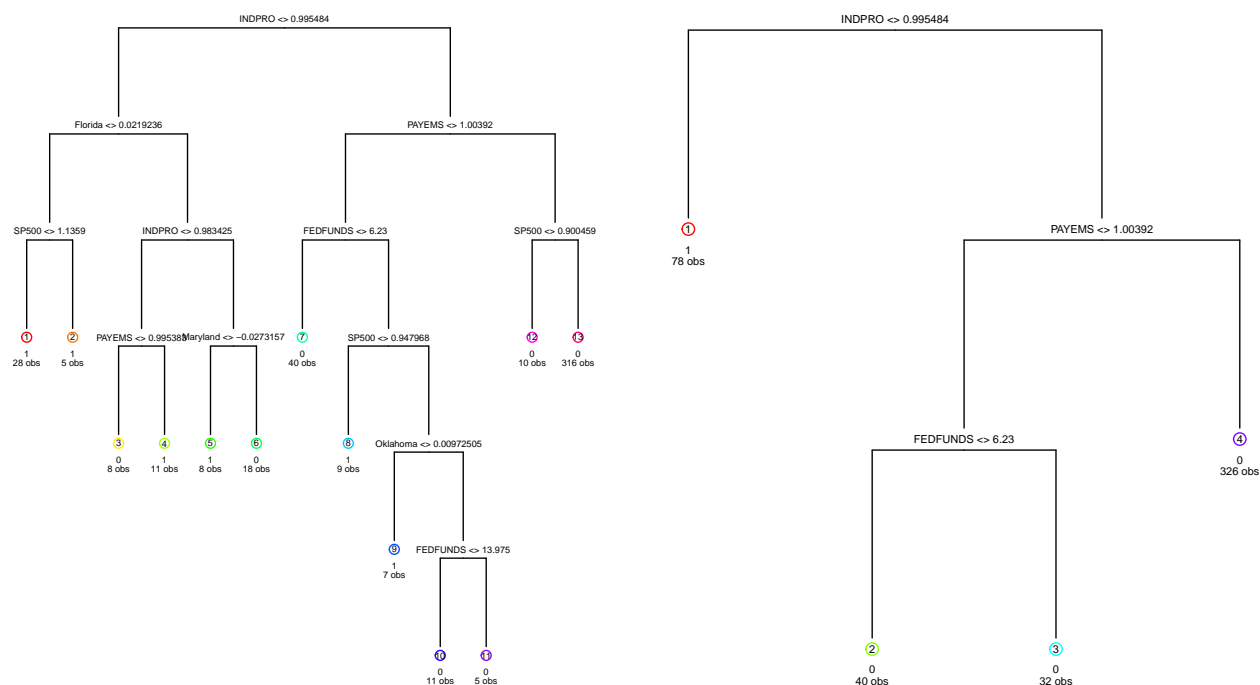

These trees were grown on the ~~training~~ data (from 1960 to 1999)
Now, we use them to predict on the ~~test~~ data (from 2000 to 2011)

Results of trees on recession data
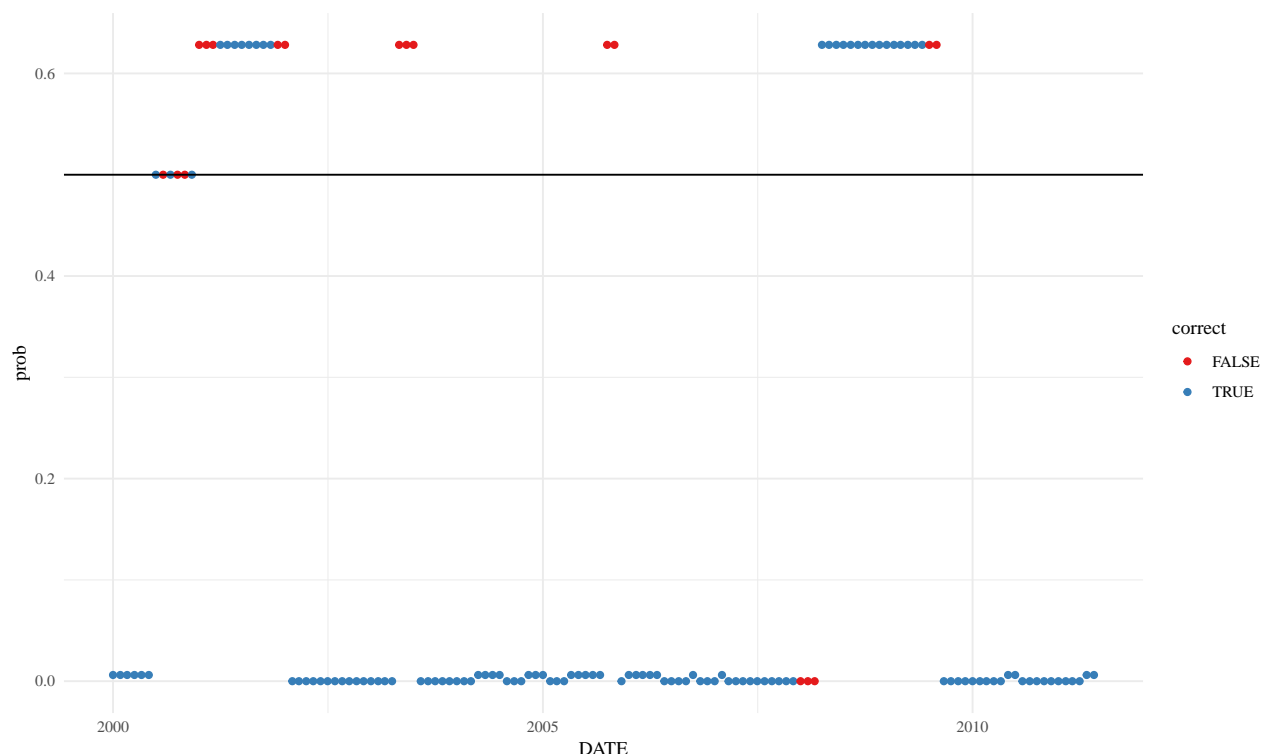
Advantages and disadvantages of trees

- Trees are very easy to explain (much easier than even linear regression).
- Some people believe that decision trees mirror human decision.
- Trees can easily be displayed graphically no matter the dimension of the data.
- Trees can easily handle qualitative predictors without the need to create dummy variables.
- Trees aren't very good at prediction.

To fix this last one, we can try to grow many trees and average their performance.

Bagging

Bagging

Many methods (trees included) tend to be designed to have lower bias but high variance.

This means that if we split the training data into two parts at random and fit a decision tree to each part, the results could be quite different.

In contrast, a low variance estimator would yield similar results if applied repeatedly to distinct data sets (consider $\widehat{f} = 0$).

Bagging, also known as bootstrap aggregation, is a general purpose procedure for reducing variance. We'll use it specifically in the context of trees, but it can be applied much more broadly.

Bagging: The main idea

Suppose we have n uncorrelated observations $Z_1, \ldots, Z_n$, each with variance σ^2.

What is the variance of

$$\overline{Z} = \frac{1}{n} \sum_{i=1}^{n} Z_i?$$

More generally, if we have B separate (uncorrelated) training sets, $1, \ldots, B$, we can form B separate model fits, $\widehat{f}^1(x), \ldots, \widehat{f}^B(x)$, and then average them:

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

Bagging: The bootstrap part

Of course, this isn't practical as having access to many training sets is unlikely. We therefore turn to the bootstrap to simulate having many training sets.

Suppose we have data $Z_1, \ldots, Z_n$ and we want to get an idea of the sampling distribution of some statistic $f(Z_1, \ldots, Z_n)$. Then we do the following

1. Choose some large number of samples, B.
2. For each $b = 1, \ldots, B$, draw a new dataset from $Z_1, \ldots, Z_n$, call it $Z_1^*, \ldots, Z_n^*$.
3. Compute $\widehat{f}_b^* = \widehat{f}(Z_1^*, \ldots, Z_n^*)$.

Bagging: The bootstrap part

Now, instead of having B separate training sets, we do B bootstrap samples. $\widehat{f}_1^*(x), \ldots, \widehat{f}_B^*(x)$, and then average them:

$$\widehat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \widehat{f}_b^*(x)$$

This process is known as Bagging (bootstrap aggregation).

Bagging trees

Bagging trees

The procedure for trees is the following

1. Choose a large number B.
2. For each $b = 1, \ldots, B$, grow an unpruned tree on the bth bootstrap draw from the data.
3. Average all these trees together.

Each tree, since it is unpruned, will have (low/high) variance and (low/high) bias.

Therefore averaging many trees results in an estimator that has lower variance and still low bias.

Caveat: Be careful bootstrapping time series data.

Bagging trees: Variable importance measures

Though bagging can improve predictive performance of trees, the trade-off is we sacrificed some interpretability.

We no longer have that nice diagram that shows the segmentation of the predictor space (or, more accurately, we have B of them).

To recover some information, we can do the following:

1. For each of the b trees and each of the p variables, we record the amount that the Gini index is reduced by the addition of that variable
2. Report the average reduction over all B trees.

Bagging trees: Variable importance measures

```r
library(randomForest)
rf = randomForest(USREC~.-DATE, data=recessions, subset=train)
predclass$rf = predict(rf, test_data)
bag = randomForest(USREC~.-DATE,data=recessions, subset=train, mtry=ncol(recessions)-2)
predclass$bag = predict(bag, test_data)
out = varImpPlot(rf)
```

```r
dfout = data.frame(variable=rownames(out), out)
ggplot(dfout[1:25,],aes(MeanDecreaseGini,reorder(variable,MeanDecreaseGini))) +
  geom_point(color=red) + ylab('variable') + xlab('Mean Decrease Gini')
```
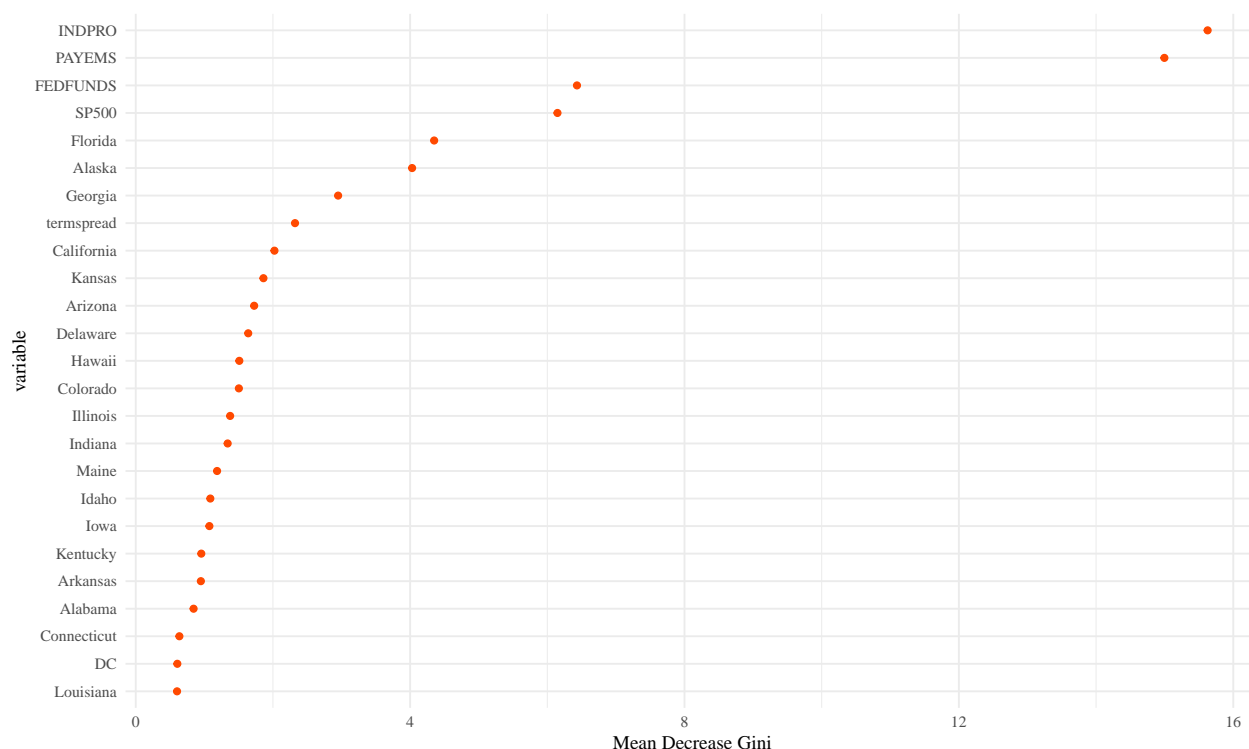

Random Forest

Random Forest is a small extension of Bagging, in which the bootstrap trees are decorrelated.

The idea is, we draw a bootstrap sample and start to build a tree.

- At each split, we randomly select m of the possible p predictors as candidates for the split.
- A new sample of size m of the predictors is taken at each split.

Usually, we use about $m = \sqrt{p}$

(this would be 7 out of 56 predictors for GDP data).

In other words, at each split, we aren't even allowed to consider the majority of possible predictors!

Random Forest

What is going on here?

Suppose there is 1 really strong predictor and many mediocre ones.

- Then each tree will have this one predictor in it,
- Therefore, each tree will look very similar (i.e. highly correlated).

- Averaging highly correlated things leads to much less variance reduction than if they were uncorrelated.

If we don't allow some trees/splits to use this important variable, each of the trees will be much less similar and hence much less correlated.

Bagging Trees is Random Forest when $m = p$, that is, when we can consider all the variables at each split.

Reporting results

There are two main ways classification results are reported:

- Sensitivity/specificity
- Confusion matrix

Reporting results: Sensitivity/specificity

~~Sensitivity:~~ The proportion of times we label 'recession', given that 'recession' is the correct answer. (True $+$)
~~Specificity:~~ The proportion of times we label 'no recession', given that 'no recession' is the correct answer. (True $-$)

We can think of this in terms of hypothesis testing. If

$$H_0 : \text{ no recession,}$$

then

~~Sensitivity:~~ $P(\text{reject } H_0 | H_0 \text{ is false})$, that is: 1 - $P(\text{Type II error})$
~~Specificity:~~ $P(\text{accept } H_0 | H_0 \text{ is true})$, that is: 1 - $P(\text{Type I error})$

Tree results: Confusion matrices

prediction	0	1
bagging		
0	107	4
1	5	22
null		
0	112	26
random forest		
0	110	6
1	2	20
tree		
0	97	3
1	15	23

Tree results: Sensitivity and Specificity

predictor	sensitivity	specificity
bag	0.846	0.955
null	NA	1.000
rf	0.769	0.982
tree	0.885	0.866

Results of Bagging on recession data

```
test_data = recessions[test,]
test_data$class = predclass$bag
test_data$prob = predict(bag, test_data, type='prob')[,2]
test_data = mutate(test_data, correct = class==USREC)
ggplot(test_data, aes(DATE, prob, color=correct)) + geom_point() +
  scale_color_brewer(palette = 'Set1') + geom_hline(yintercept = .5)
```
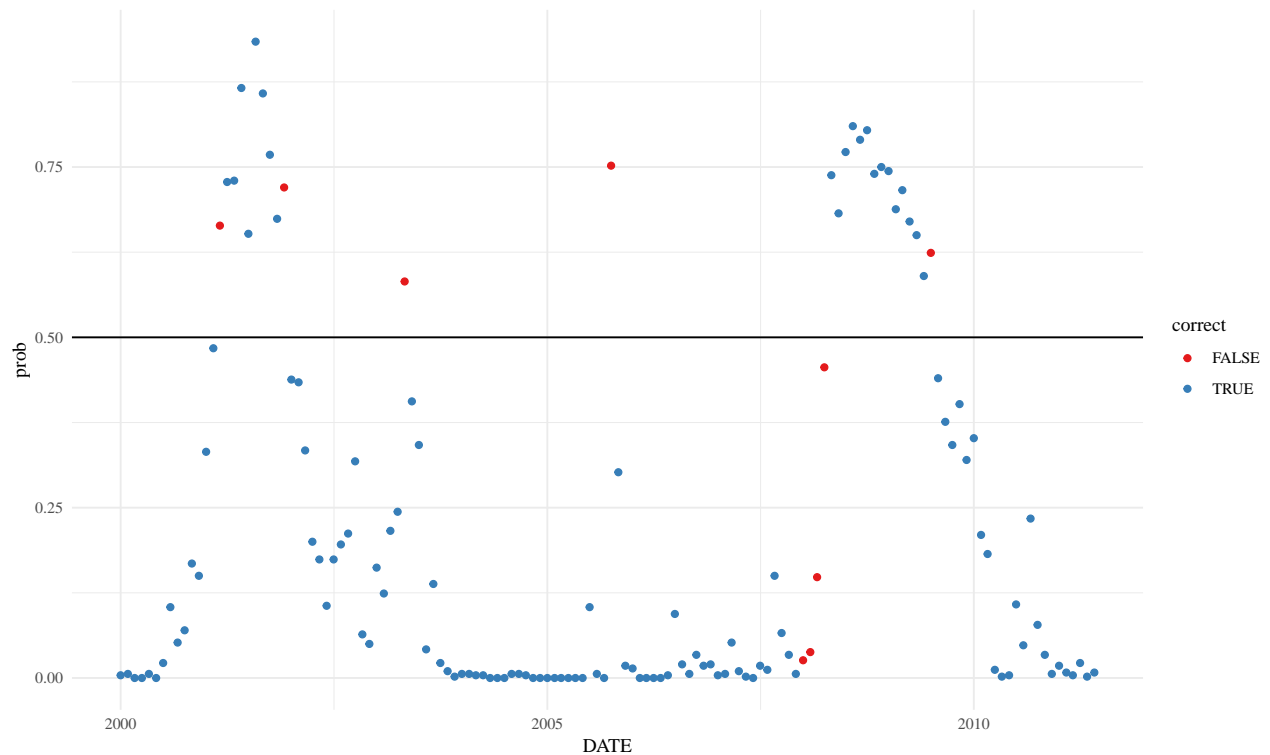
Out-of-Bag error estimation (OOB)

One can show that, on average, drawing n samples from n observations with replacement (also known as bootstrap) results in about 2/3 of the observations being selected.

The remaining one-third of the observations not used are referred to as ~~out-of-bag (OOB)~~.

We can think of it as a for-free cross-validation.

Each time a tree is grown, we can get its prediction error on the unused observations. We average this over all bootstrap samples.

Out-of-bag error estimation for bagging

```
tab = table(predict(bag),recessions$USREC[train])
kable(tab)
```

	0	1
0	395	21
1	14	46

```
sum(diag(tab))/sum(tab)
```

```
## [1] 0.9264706
```

Classification recap

There are many, many different classification algorithms.

Different methods will work bettter in different situations.

~~Linear:~~
Logisitic regression,
linear discriminant analysis (LDA),
GLMNET, separating hyperplanes

~~Non-linear:~~
quadratic discriminant analysis (QDA),
trees (and associated methods),
K-nearest neighbors (KNN)
Support vector machines (SVM)
Neural networks