Today (week 10) in IST 380

Claremont International Fair ~ Sat.

Selecting features + models

and snacks!

Deliverable Schedule…

Homework #8 (maps & project, part 1) is graded…

Project, part 2 is due this Saturday (4/20)

Clustering results from your data set (or subsets)

- Can start on model-testing (but it's optional)

Project talks will be on **Monday** (4/29)

- Include model-selection and testing…
- Include initial results from a predictive model

10-15 minute talk *(example next week…)*

Extra: *"Green-screening" your image…*

Project schedule

4/8: analyzing time series

Due 4/13: choosing the dataset

- Explain the dataset
- Initial set of questions about it
- A desired predictive model
- Two initial visualizations (in R) (read the data!)
- Run an initial SVM, NN, and k-NN classifier
- Other systems OK, but only *in addition*

4/15: model selection and gradient-boosted classifiers

Due 4/20: feature selection and exploration

2nd write-up due on 4/20

- Define features/subsets of your data
- Try the unsupervised algorithms
  hierarchical and k-means clustering
- Visualize and summarize the results

4/22: example final project talk / short session

"*Due*" 4/27: model selection / predictive modeling

No need to submit a 3rd write-up until the final draft. Instead, include the results in your talk…

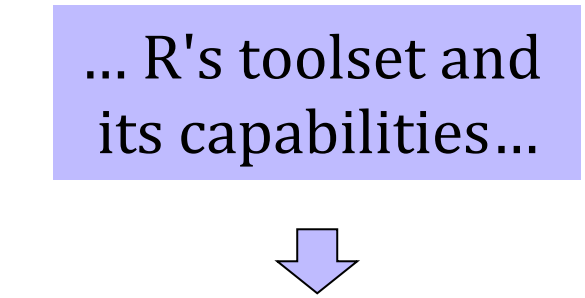
- Decide on a cross-validation strategy
  10-fold, models to check, parameters to test…
- Try out appropriate algorithms
  linear and/or logistic regression, decision trees, random forests, SVMs, NNs, and k-NNs, time series, ***TreeBoost***
- Run them and summarize the results
- Create a predictive model (or ensemble)

On 4/29: final-project talks

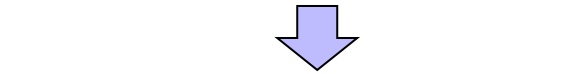
Final project report due 5/11/2013 (No class 5/6/2013)

R path!

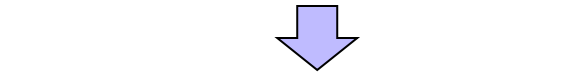
… R's toolset and its capabilities…



feature-selection functions (and images)



cross validation



TreeBoost models…

We've been pulled in!

Last time: *Project datasets*

Payal:	movie recommendation data
Joe/Suleng:	stock data?
Lauren:	NYC school performance data
Jordan:	Yelp! dataset!
Obay:	Parallel medical texts
Kareem:	Twitter data

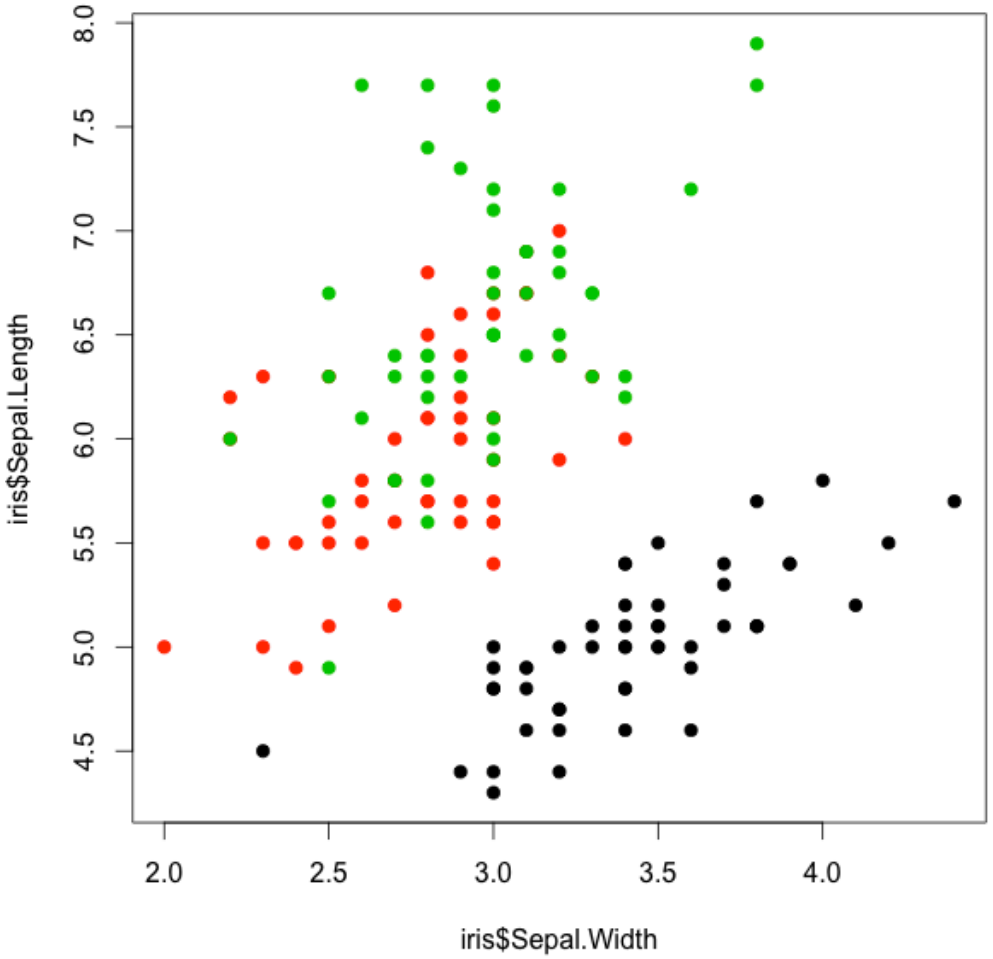
Clustering analyses…

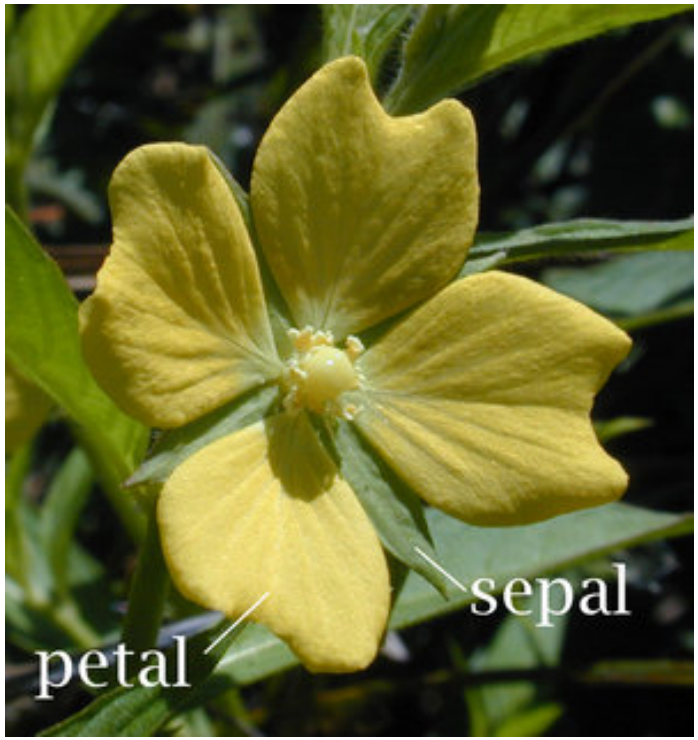
- decide on – or create – features of interest
- include at least *two sets* of feature combinations, i.e., run two analyses
- plot the results of hierarchical clustering and k-means
- if you do have labels (most do) – then compare your clustering with the labels!
- natural transition to modeling, for example, with k-nearest neighbors *(this is optional…)*

Clustering examples…

iris dataset

- decide on – or create – features of interest



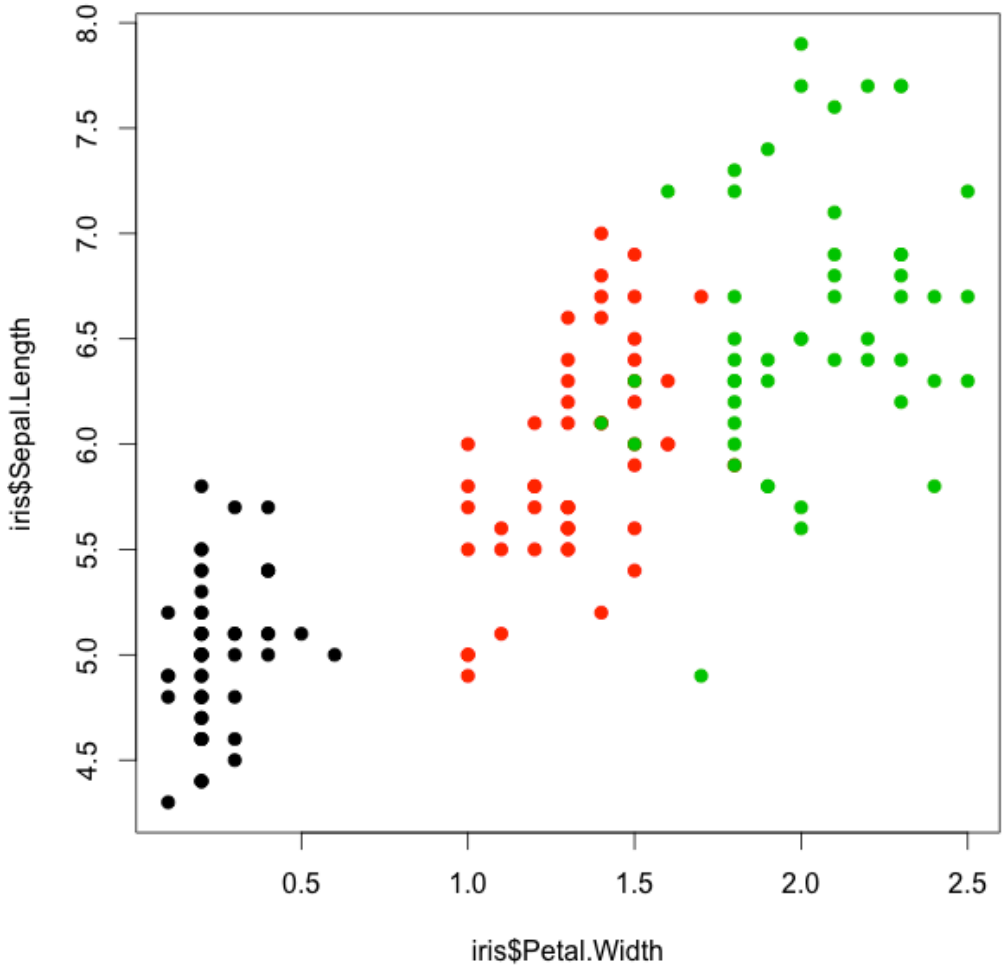


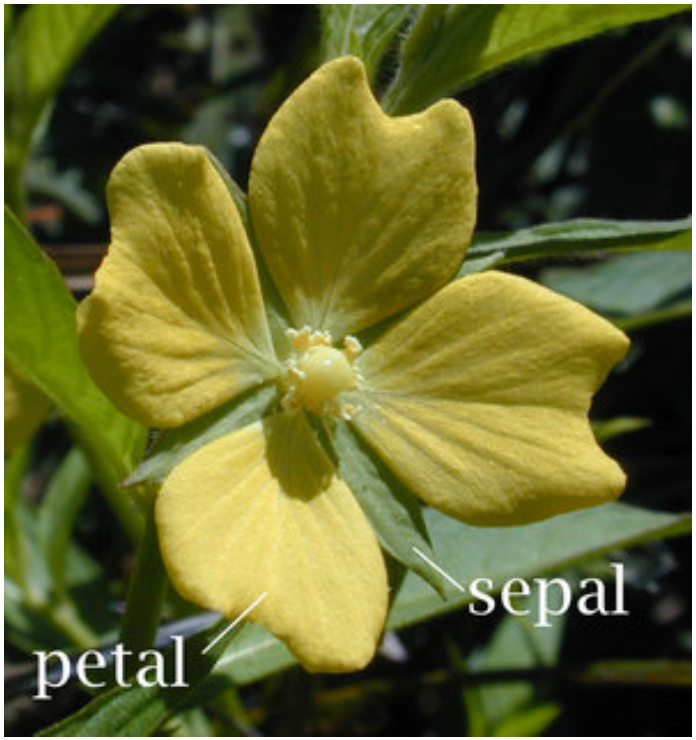
Sepal length vs. width?

Clustering examples…

iris dataset

- decide on – or create – features of interest





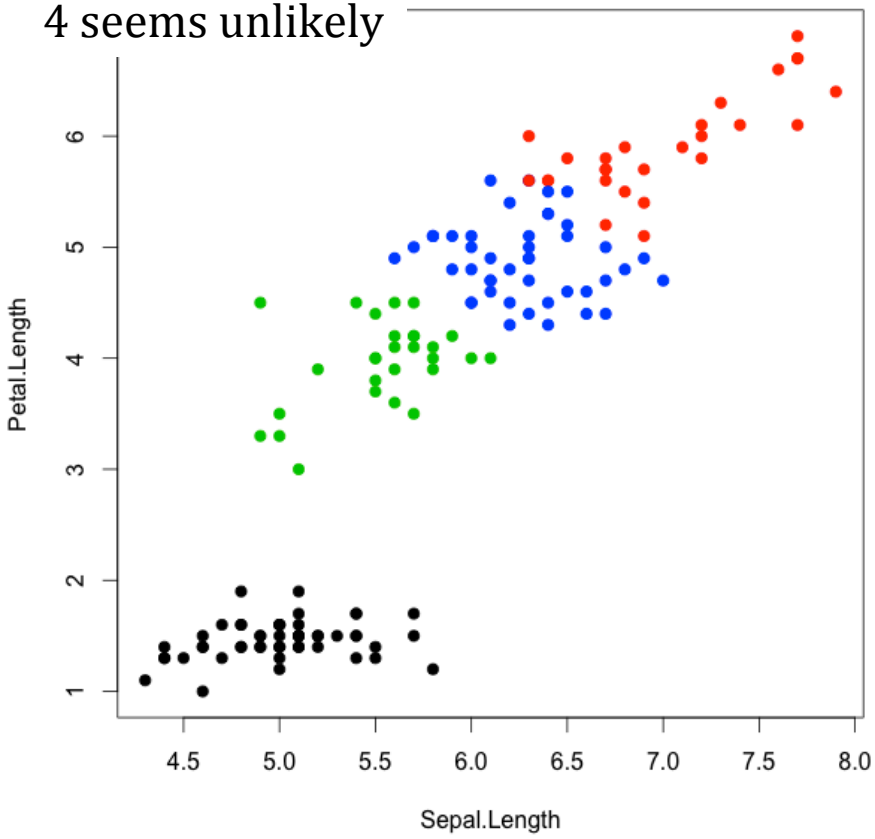
Sepal length vs. Petal.Width?

Clustering examples…

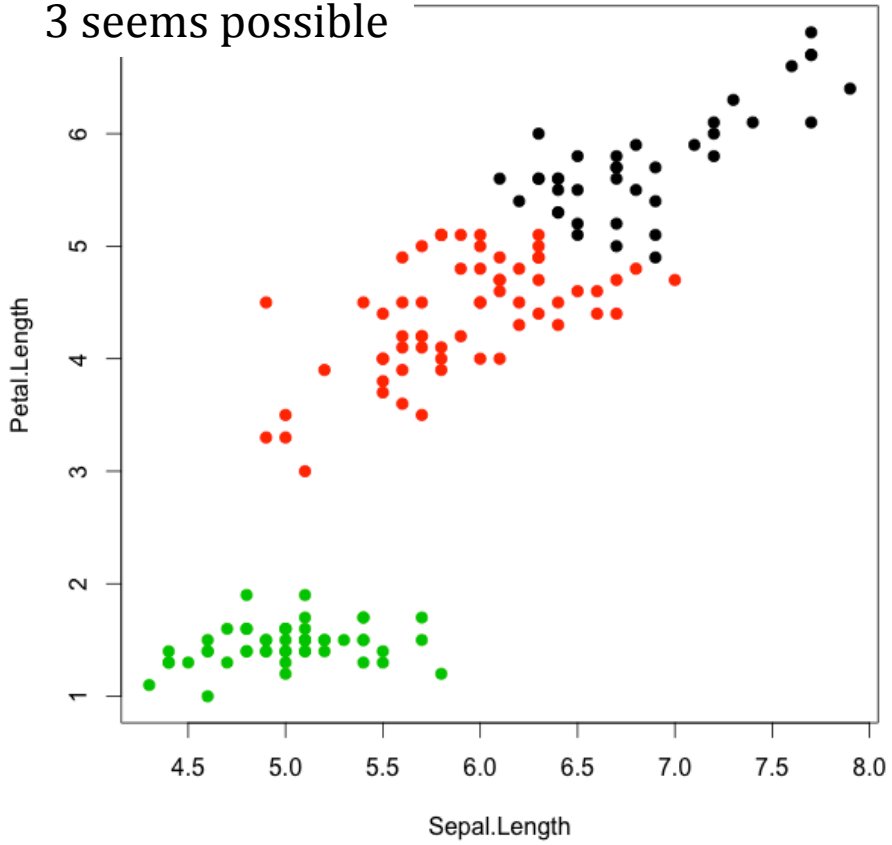
iris dataset

- include at least *two sets* of feature combinations, i.e., run two analyses

4 seems unlikely



3 seems possible



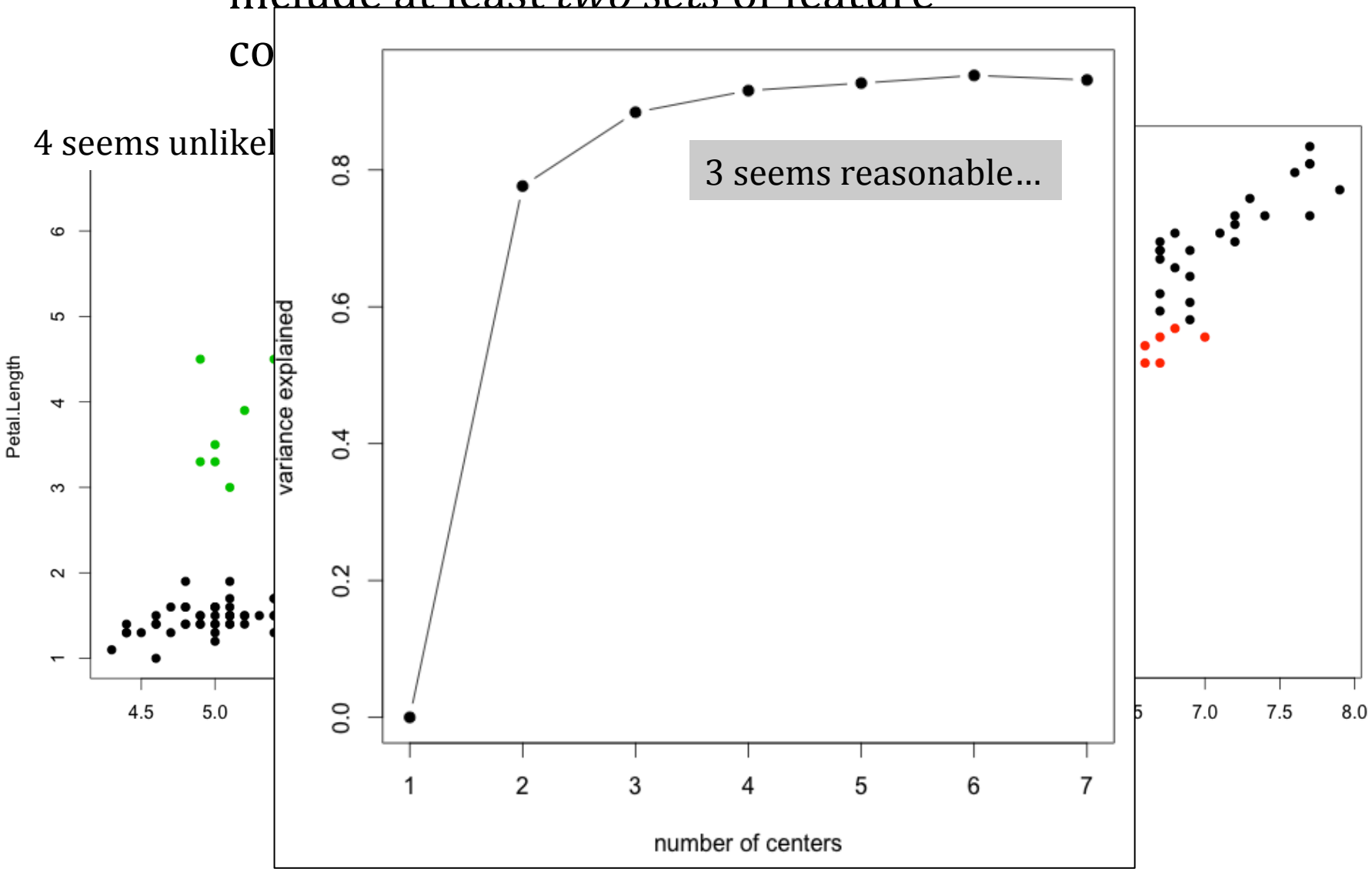
number of clusers – for all features

Clustering examples…

iris dataset

- include at least *two sets* of feature co

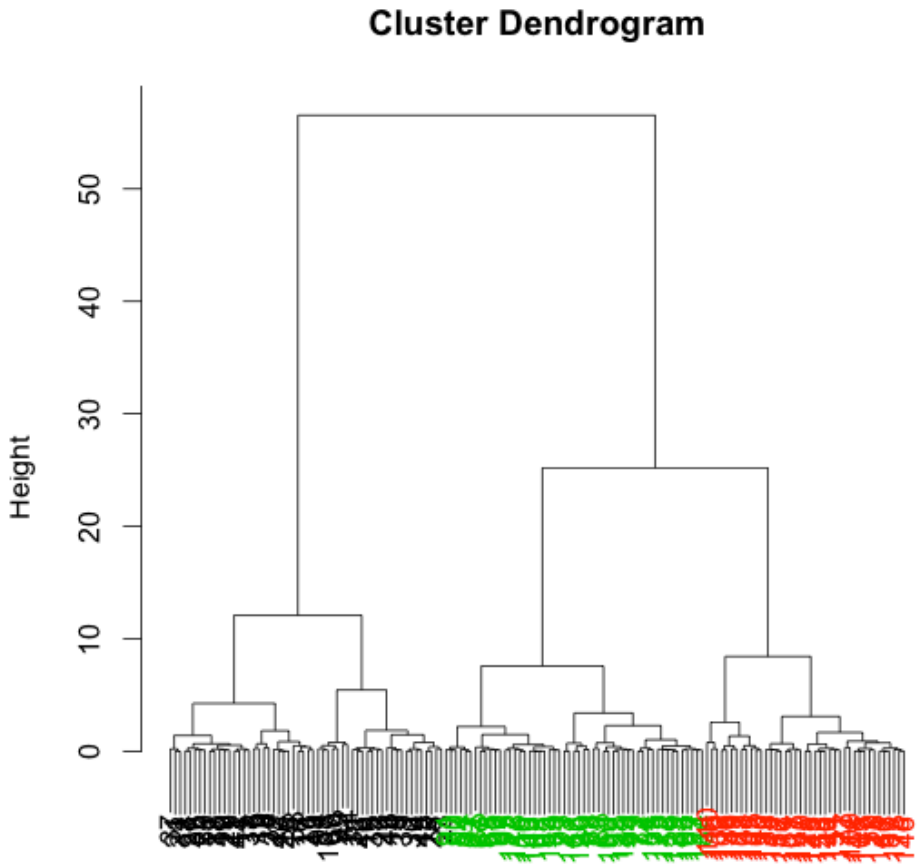
4 seems unlikel

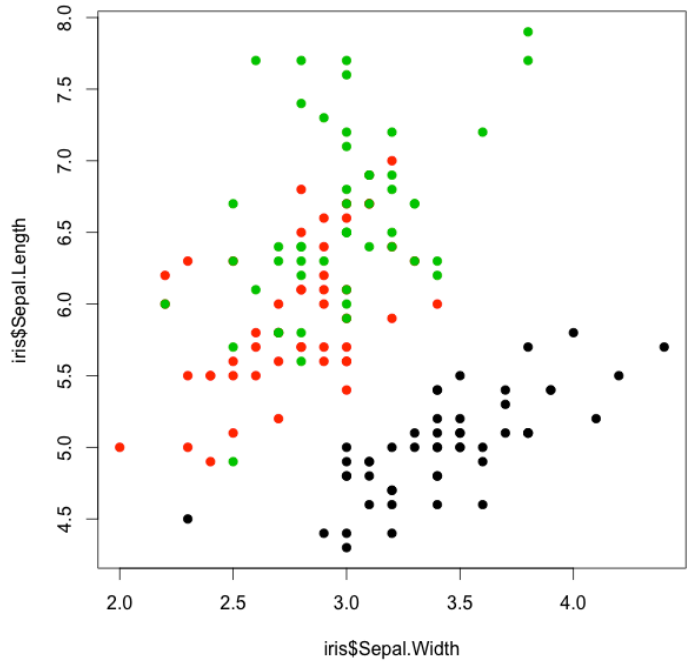


Clustering examples…

iris dataset

- plot the results of hierarchical clustering and k-means



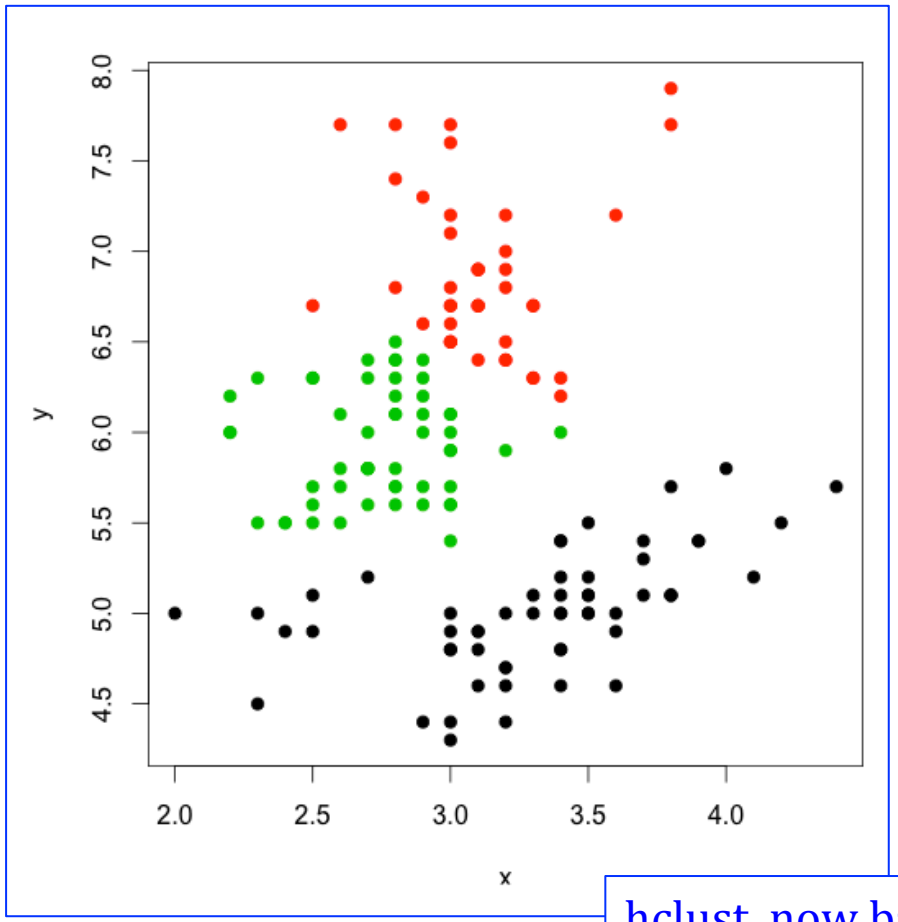


Sepal length vs. width?

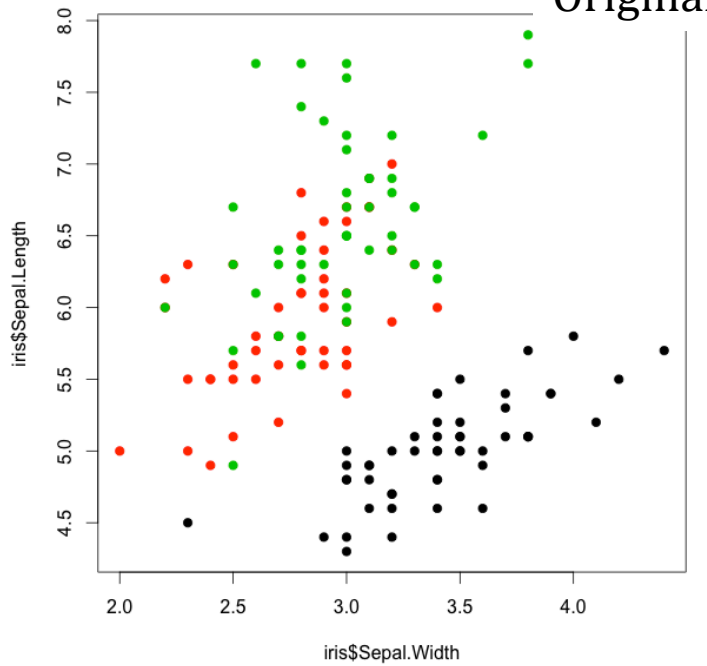
Clustering examples…

iris dataset

- plot the results of hierarchical clustering and k-means



Original



hclust, now back in the original space…

not ideal…

Clustering examples…

iris dataset

```
> clusters
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [38] 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 3 3 3 2 1 2 1 1 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3
 [75] 3 2 2 2 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 1 3 3 3 3 1 3 2 3 2 3 2 2 1 2 2 2 2 2 2
[112] 3 2 3 3 2 2 2 2 3 2 3 2 3 2 2 3 3 3 2 2 2 3 3 3 2 2 2 3 2 2 2 2 3 2 2 2 3 2 2
[149] 2 3
```

Aargh – the labels don't match…

- if you do have labels (most do) – then compare your clustering with the labels!

Clustering examples…

iris dataset

```
> clusters
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 3 3 3 2 1 2 1 1 3 3 3 3 2 3 3 3 3 3 3 3 3
 [75] 3 2 2 2 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 1 3 3 3 3 1 3 2 3 2 3 2 2 1 2 2 2 2
[112] 3 2 3 3 2 2 2 2 3 2 3 2 3 2 2 3 3 3 2 2 2 3 3 3 2 2 2 [illegible] [illegible] 2 2 3 2 2 2 3 2
[149] 2 3
```

Aargh – the labels don't match…

```
> c24 <- ifelse(clusters==2,4,clusters)
> c24
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 4 4 4 3 3 3 4 1 4 1 1 3 3 3 3 4 3 3 3 3 3 3 3 3
 [75] 3 4 4 4 3 3 3 3 3 3 3 3 4 3 3 3 3 3 3 1 3 3 3 3 1 3 4 3 4 3 4 4 1 4 4 4 4
[112] 3 4 3 3 4 4 4 4 3 4 3 4 3 4 4 3 3 3 4 4 4 3 3 3 4 4 4 3 4 4 4 3 4 4 4 3 4
[149] 4 3
```

```
> c32 <- ifelse(c24==3,2,c24)
>
> c32
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 4 4 4 2 2 2 4 1 4 1 1 2 2 2 2 4 2 2 2 2 2 2 2 2
 [75] 2 4 4 4 2 2 2 2 2 2 2 2 4 2 2 2 2 2 2 1 2 2 2 2 1 2 4 2 4 2 4 4 1 4 4 4 4
[112] 2 4 2 2 4 4 4 4 2 4 2 4 2 4 4 2 2 2 4 4 4 2 2 2 4 4 4 2 4 4 4 2 4 4 4 2 4
[149] 4 2
```

```
> c_final <- ifelse(c32==4,3,c32)
> c_final
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 2 2 2 3 1 3 1 1 2 2 2 2 3 2 2 2 2 2 2 2 2
 [75] 2 3 3 3 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 1 2 2 2 2 1 2 3 2 3 2 3 3 1 3 3 3 3
[112] 2 3 2 2 3 3 3 3 2 3 2 3 2 3 3 2 2 2 3 3 3 2 2 2 3 3 3 2 3 3 3 2 3 3 3 2 3
[149] 3 2
```

- if you do have labels (most do) – then compare your clustering with the labels!

Rename the labels a couple of times…

or a small function will do that for you – here I did it by hand…

Clustering examples…

iris dataset

```
> clusters
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [38] 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 3 3 3 2 1 2 1 1 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 [75] 3 2 2 2 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3 1 3 3 3 3 1 3 2 3 2 3 2 2 1 2 2 2 2 2 2 2 2 2
[112] 3 2 3 3 2 2 2 2 3 2 3 2 3 2 2 3 3 3 2 2 2 3 3 3 2 2 2 3 2 2 2 3 2 2 2 3 2 2 2 3 2 2 2
[149] 2 3
```

```
> c24 <- ifelse(clusters==2,4,clusters)
> c24
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [38] 1 1 1 1 1 1 1 1 1 1 1 1 4 4 4 3 3 3 4 1 4 1 1 3 3 3 3 4 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 [75] 3 4 4 4 3 3 3 3 3 3 3 3 4 3 3 3 3 3 3 3 1 3 3 3 3 1 3 4 3 4 3 4 4 1 4 4 4 4 4 4 4 4 4
[112] 3 4 3 3 4 4 4 4 3 4 3 4 3 4 4 3 3 3 4 4 4 3 3 3 4 4 4 3 4 4 4 3 4 4 4 3 4 4 4 3 4 4 4
[149] 4 3
```

```
> c32 <- ifelse(c24==3,2,c24)
>
> c32
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [38] 1 1 1 1 1 1 1 1 1 1 1 1 4 4 4 2 2 2 4 1 4 1 1 2 2 2 2 4 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [75] 2 4 4 4 2 2 2 2 2 2 2 2 4 2 2 2 2 2 2 2 1 2 2 2 2 1 2 4 2 4 2 4 4 1 4 4 4 4 4 4 4 4 4
[112] 2 4 2 2 4 4 4 4 2 4 2 4 2 4 4 2 2 2 4 4 4 2 2 2 4 4 4 2 4 4 4 2 4 4 4 2 4 4 4 2 4 4 4
[149] 4 2
```

```
> c_final <- ifelse(c32
> c_final
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [38] 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 2 2 2 3 1 3 1 1 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [75] 2 3 3 3 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 1 2 2 2 2 1 2 3 2 3 2 3 3 1 3 3 3 3 3 3 3 3 3
[112] 2 3 2 2 3 3 3 3 2 3 2 3 2 3 3 2 2 2 3 3 3 2 2 2 3 3 3 2 3 3 3 2 3 3 3 2 3 3 3 2 3 3 3
[149] 3 2
```

```
> sum( c_final != as.numeric(iris$Species) )
[1] 34
> 34/150.0
[1] 0.2266667
```

- if you do have labels (most do) – then compare your clustering with the labels!

probably not the strongest match…

Clustering examples…

iris dataset

- natural transition to modeling, for example, with k-nearest neighbors *(this is optional…)*

We'll look at model selection and verification today…

Feature *definition*…

depending on your dataset, you may benefit from carefully defining meaningful features…

Examples:

lexical density feature (Obay mentioned this…)

cool/useful/funny (Yelp's dataset)

socioeconomic status (NYC schools dataset)

"rating context" (movie-ratings dataset)

Feature *definition*…

depending on your dataset, you may benefit from carefully defining meaningful features…

Examples:

lexical density feature (Obay mentioned this…)

cool/useful/funny (Yelp's dataset)

socioeconomic status (NYC schools dataset)

"rating context" (movie-ratings dataset)

often: write-your-own (or gather-your-own) functions + apply them

Feature-defining…

Example from Payal's movie dataset…

100,000 rows
don't use them all until you need to!

user#	movie#	rating
196	42	5
301	16	2
55	101	5
4	228	5
196	15	4
301	42	1

Write a small function…

Example from Payal's movie dataset…

100,000 rows
don't use them all until you need to!

user#	movie#	rating
196	42	5
301	16	2
55	101	5
4	228	5
196	15	4
301	42	1

pseudocode!

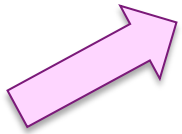
```
get_user_ave <- function( user# )
{
  return( average of user#'s ratings )
}
```

Add new columns

Example from Payal's movie dataset…

100,000 rows
don't use them all until you need to!

user#	movie#	rating	**user_ave**
196	42	5	4.5
301	16	2	1.5
55	101	5	3.1
4	228	5	4.9
196	15	4	4.5
301	42	1	1.5



pseudocode!

```
get_user_ave <- function( user# )
{
  return( average of user#'s ratings )
}
```

"SQL queries," all pre-fetched. ***Others?***

Pixel-features…

```
# may need to use getwd() and setwd("dir") to see the image
ar <- readJPEG("zach10p.jpg")
# ar is an array of pixels, i.e., an image
plot.new()
# we rotate the image so that it looks correct
art <- transpose(ar)
# note that this might seem to imply "art"
# but it's far from art!
HT <- dim(art)[1]  # we'll use these globals...
WD <- dim(art)[2]  # and for width...
CH <- dim(art)[3]  # n channels
# may or may not need interpolate=F
# see ?readPNG for more image handling
rasterImage( art, 0, 0, 1, 1,  interpolate=F )
```

image-handling in R

need a ***function*** to distinguish green from everything else…

Pixel-features…

getting a single pixel…

```
> get_rgb(art,1)
    x   y         r         g         b
1 294 268 0.2313725 0.4352941 0.1529412
```

out of 1.0 (also, often 0 to 255)

low-res: 400x300

need a ***function*** to distinguish green from everything else…

Pixel-feature examples…

```
# crop so that it's a true green-screen
art <- art[1:270,,]
rasterImage( art, 0, 0, 1, 1,  interpolate=F )
```

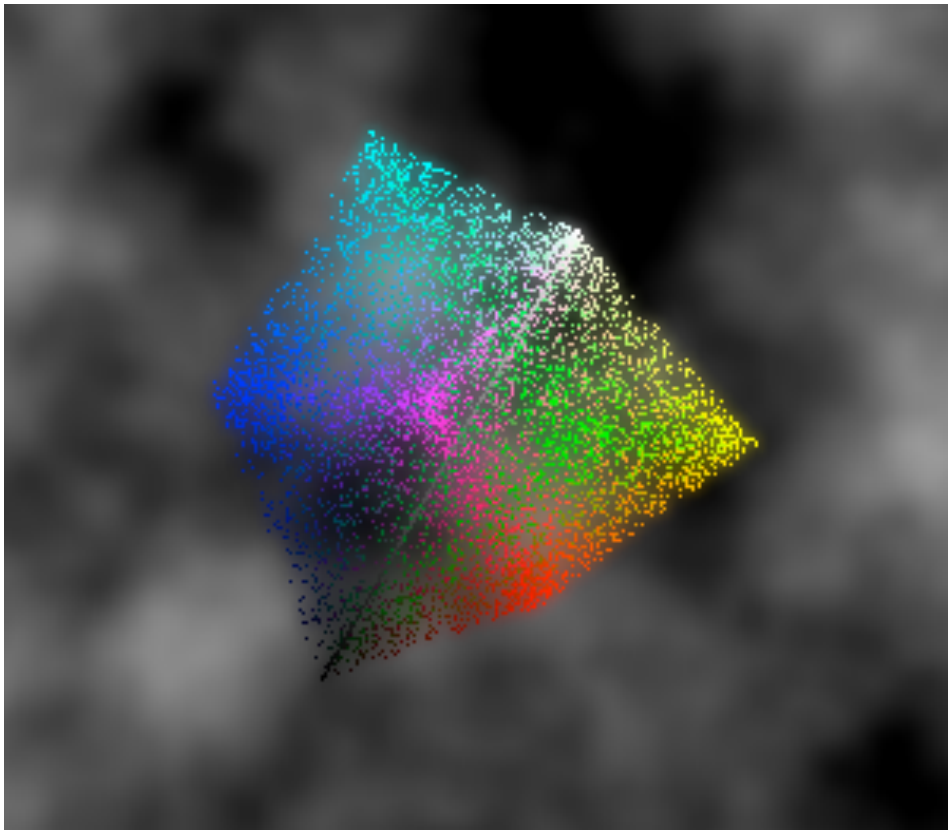
What's green?

Look at many pixels…

Give examples & learn…

Pixel-feature examples…

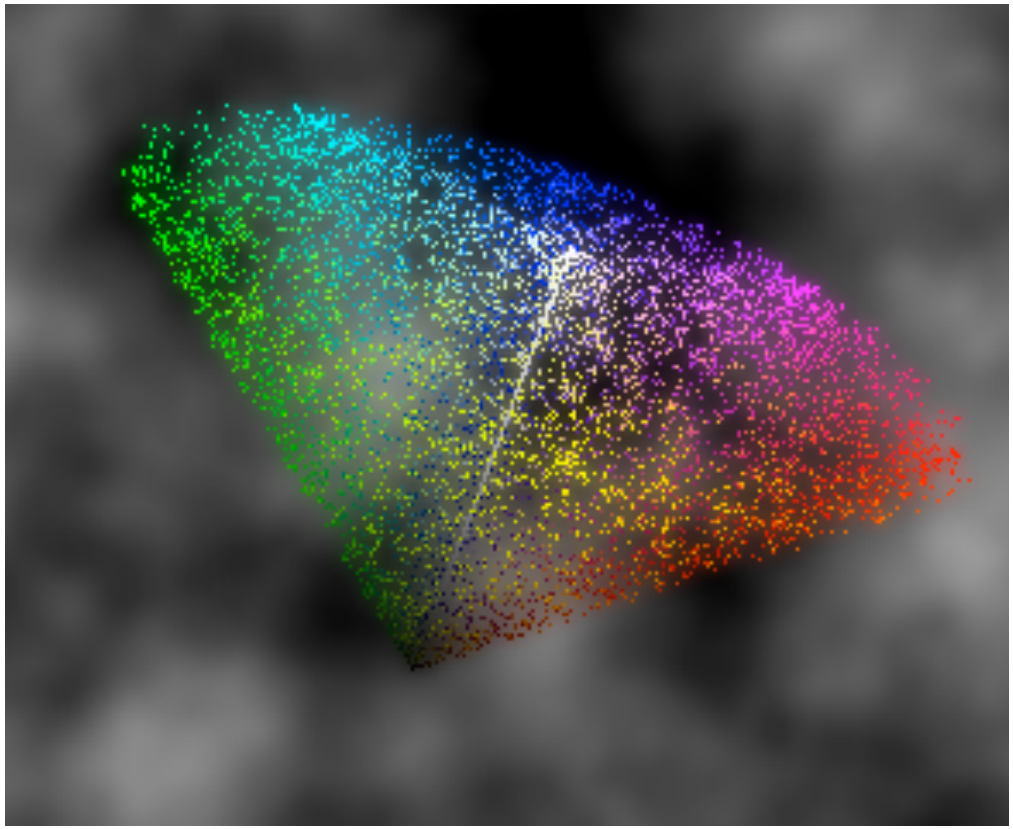
RGB is not an ideal representation for defining a single "color"



www.flashandmath.com/advanced/color/

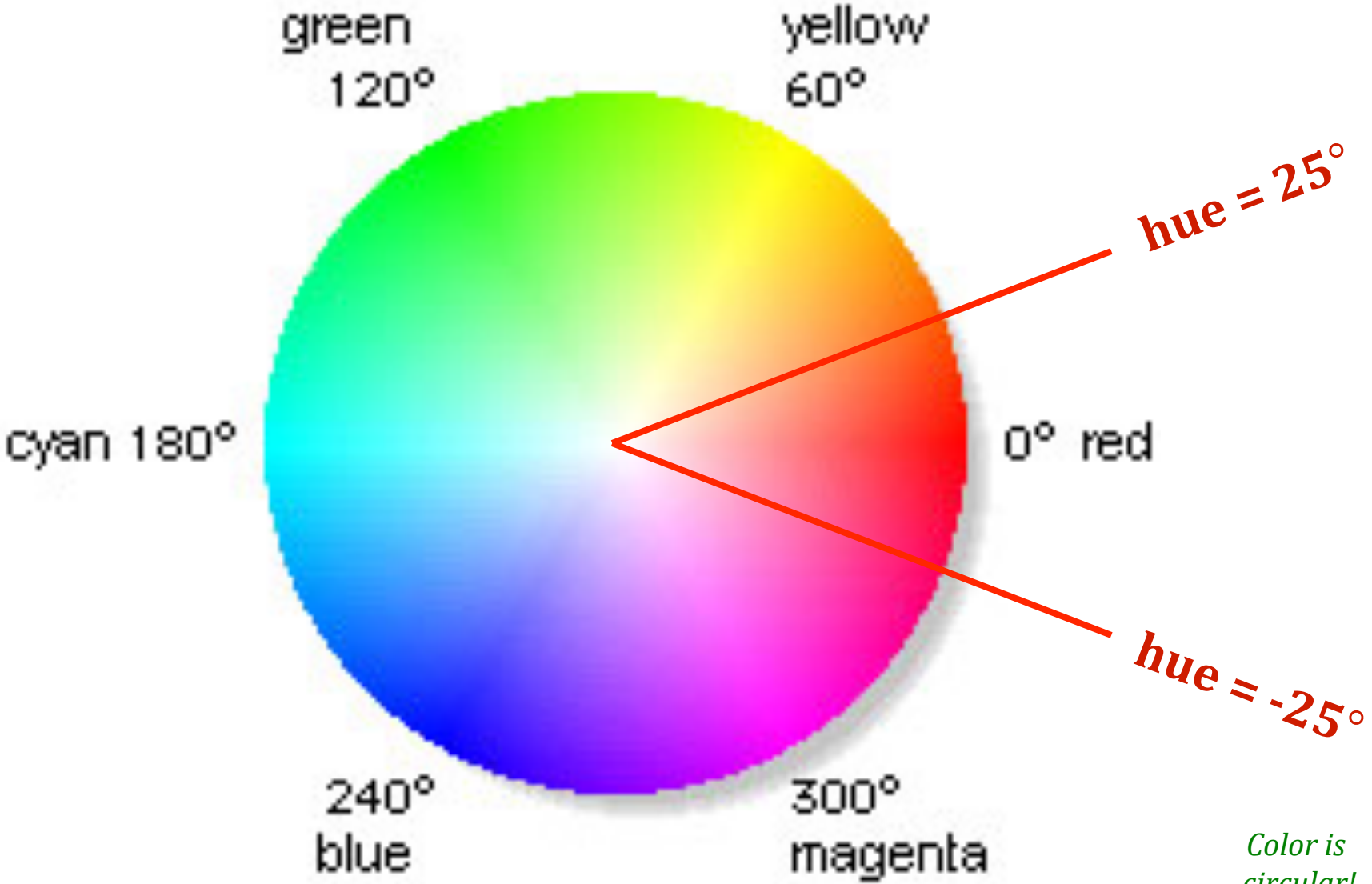
Pixel-feature examples…

HSV tips the RGB cube "on its axis" and converts it to a ***cone***



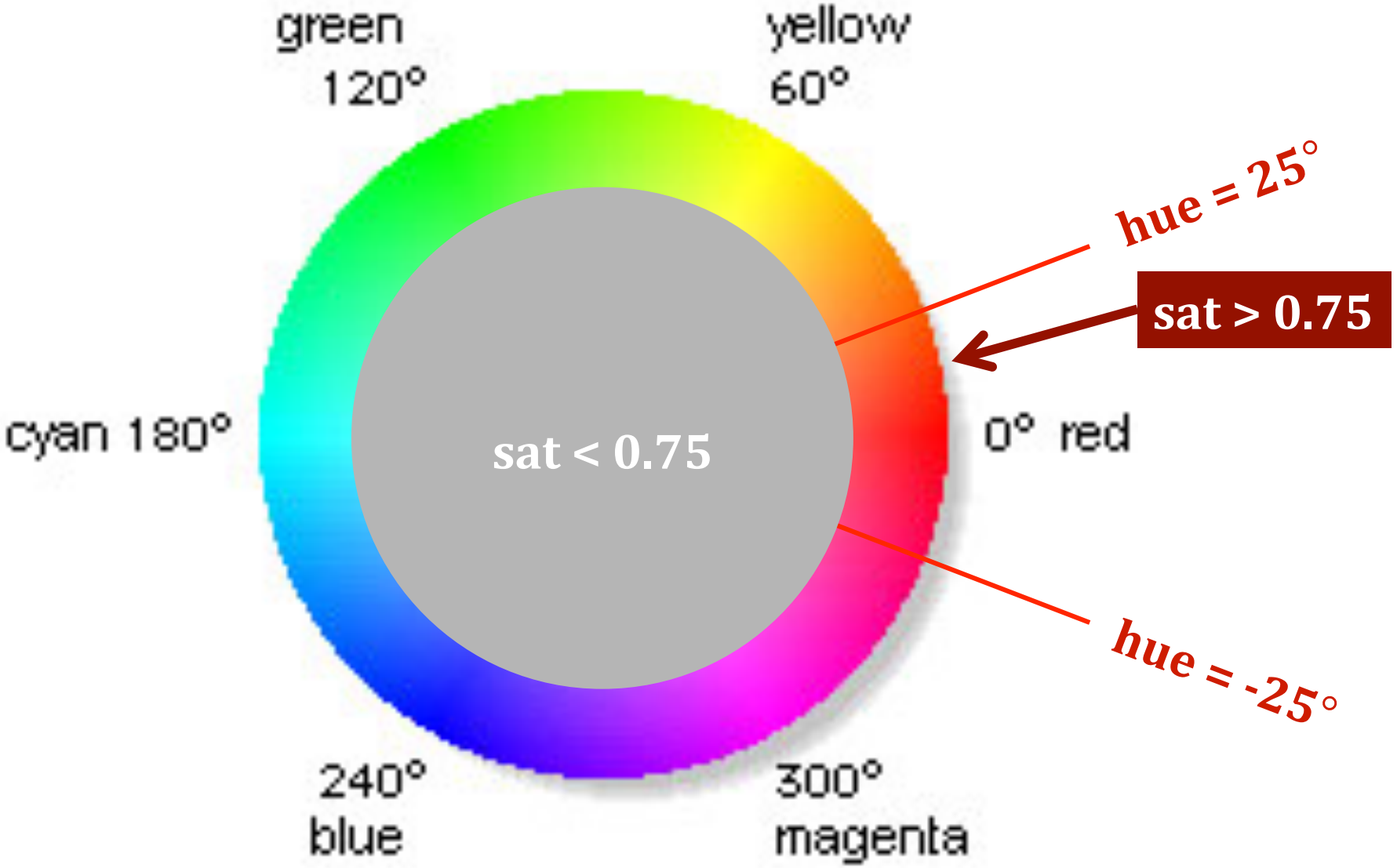
You don't have to invent your features – use well-established ones (if they exist)!

Hue ~ color ***name***



Color is circular!

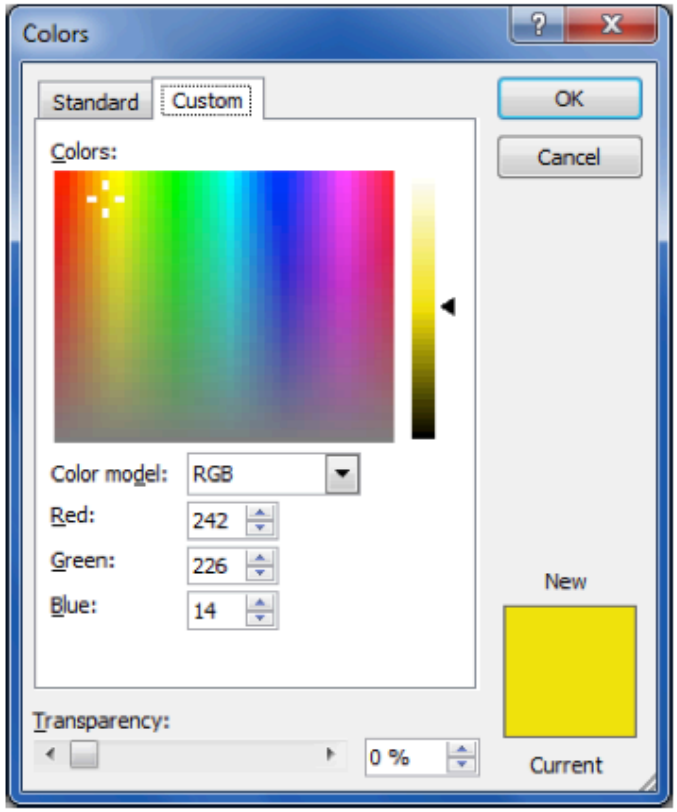
Saturation is color ***intensity***



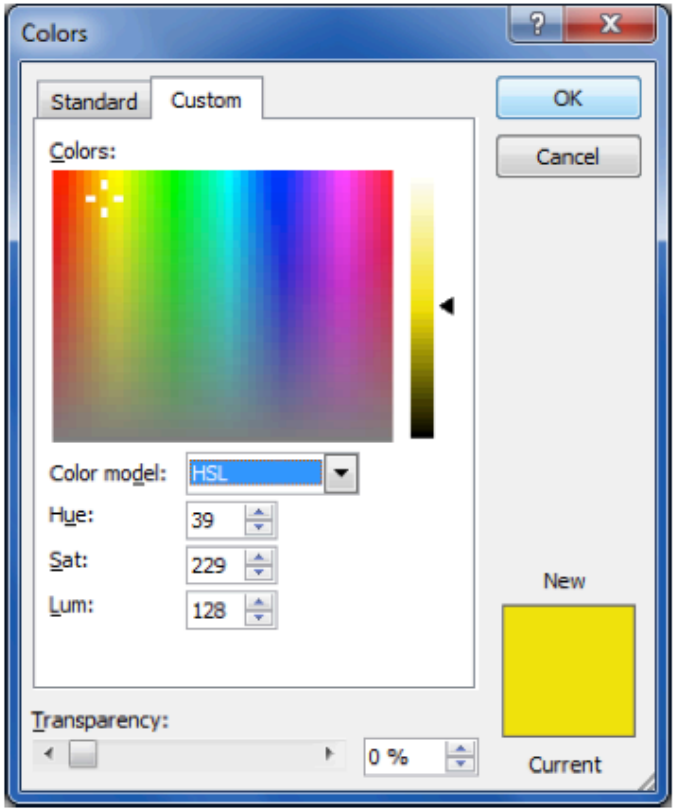
Color choosers…

One pixel

***Red, Green, Blue** values*

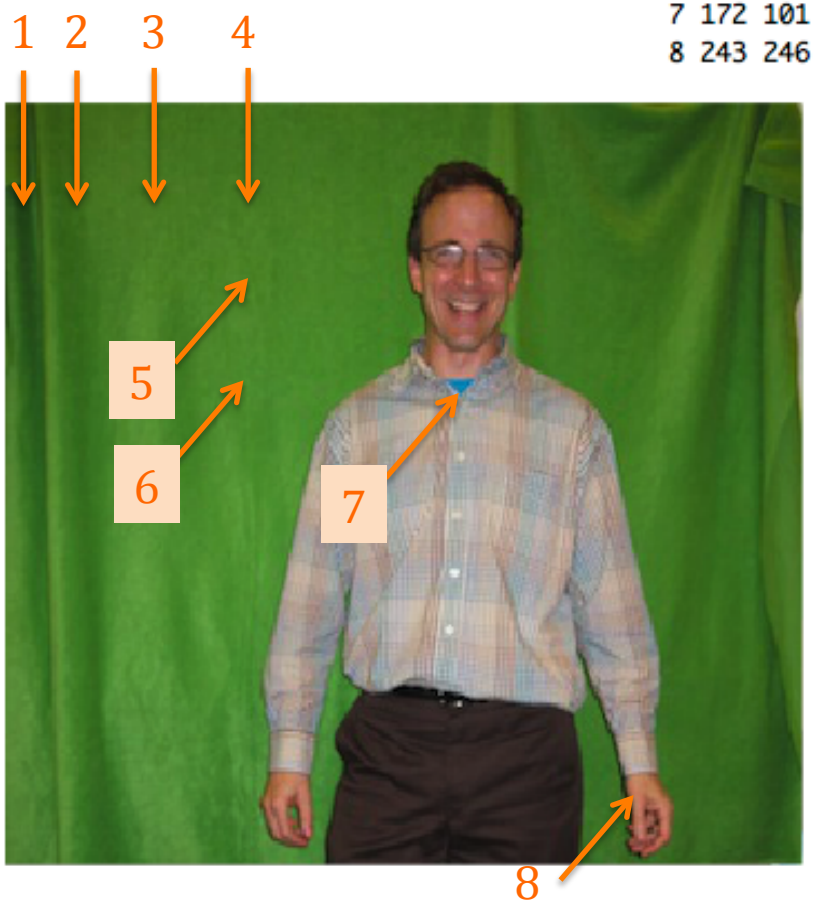


***Hue, Sat, Value** values*



Pixel-feature examples…

```
> get_all(art,8)
    x   y         r         g         b         h         s         v
1   6  35 0.1568627 0.3254902 0.1176471 0.30188679 0.6385542 0.3254902
2  25  36 0.2784314 0.4745098 0.1411765 0.26470588 0.7024793 0.4745098
3  46  36 0.3529412 0.5490196 0.2078431 0.26245211 0.6214286 0.5490196
4  71  35 0.3411765 0.5490196 0.1960784 0.26481481 0.6428571 0.5490196
5  70  59 0.3450980 0.5607843 0.2039216 0.26739927 0.6363636 0.5607843
6  70  84 0.3333333 0.5686275 0.2156863 0.27777778 0.6206897 0.5686275
7 172 101 0.1529412 0.6078431 0.7686275 0.54352442 0.8010204 0.7686275
8 243 246 0.7686275 0.5529412 0.4313725 0.06007752 0.4387755 0.7686275
```



What's green?

Look at many pixels…

Give examples & learn…

RGB

original

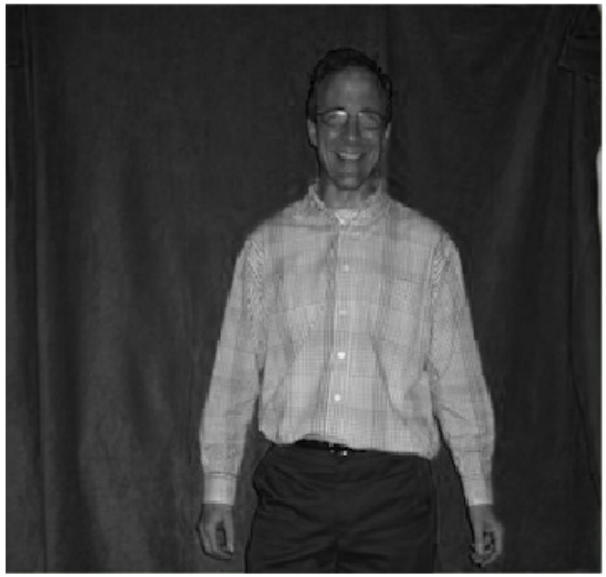
The original data, shown as different image bands in grayscale:

Why are these… ?

red

green

blue



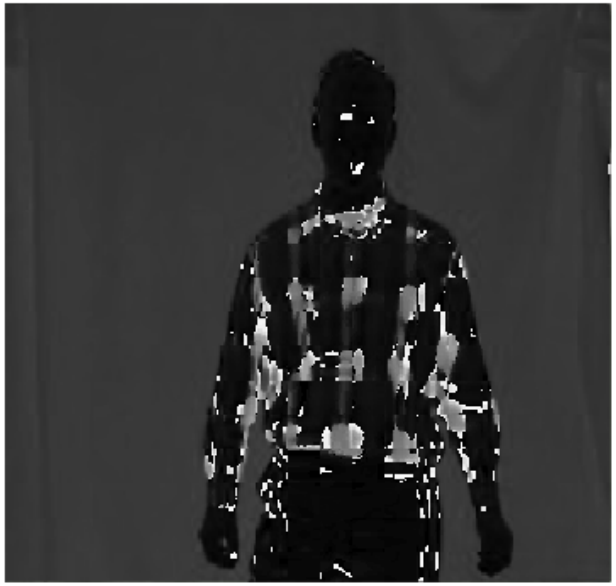
HSV

original

The original data, shown as different image bands in grayscale:

Why are these… ?

hue



sat

value

HSV together…

The R calls that created the previous slides' grayscale images:

```
> im <- get_ch( art, "rgb" ) ; rasterImage( im, 0, 0, 1, 1,  interpolate=F )
> im <- get_ch( art, "r" ) ; rasterImage( im, 0, 0, 1, 1,  interpolate=F )
> im <- get_ch( art, "g" ) ; rasterImage( im, 0, 0, 1, 1,  interpolate=F )
> im <- get_ch( art, "b" ) ; rasterImage( im, 0, 0, 1, 1,  interpolate=F )
> im <- get_ch( art, "h" ) ; rasterImage( im, 0, 0, 1, 1,  interpolate=F )
> im <- get_ch( art, "s" ) ; rasterImage( im, 0, 0, 1, 1,  interpolate=F )
> im <- get_ch( art, "v" ) ; rasterImage( im, 0, 0, 1, 1,  interpolate=F )
> im <- get_ch( art, "hsv" ) ; rasterImage( im, 0, 0, 1, 1,  interpolate=F )
```

See if you can identify which is which:

Red, Green, Blue, Hue, Saturation, Value (Lightness)


SPAM
Classic
NET WT
340g
Hormel
Foods


SPAM
SPAM

Green Screen (red)

didn't catch all of the green…

better!

Green Screen (red)

a start…

```
# this is the test for green!
GREEN <- (df$hue_row > .15)  # but it's definitely not complete!!

df[which(GREEN),1] <- 1.0
df[which(GREEN),2] <- 0.0
df[which(GREEN),3] <- 0.0

cat("len of whichGREEN ", length(which(GREEN)))
```

Background…


STARBUCKS COFFEE
STARBUCKS COFFEE
© 2008 Small World Photos

Green Screen

see-through shirt!

better – but not perfect!

hair needs some work!

Raw material..

Raw ma

Pixel-feature examples…

Let's see about this…

… maybe next time.

Model selection

- split-sample
- cross-validation
- bootstrap validation

Model selection

- split-sample

Set aside a portion of your data (~30%) for testing.

Use the rest for training…

Not typical except when the TEST data is collected later…

Model selection

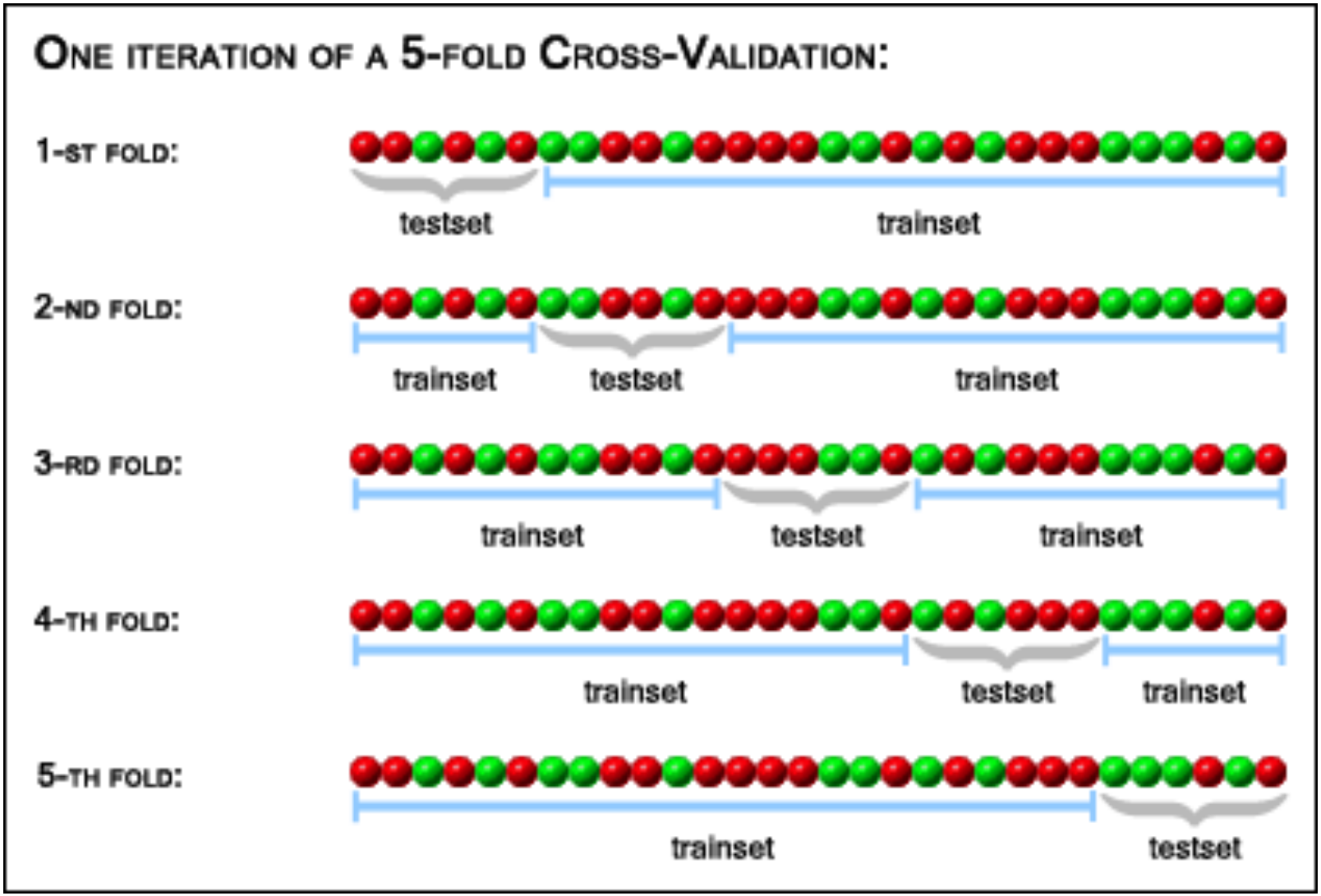
- cross-validation

Use split-sample testing…

…with many possible splits!

CV operation…

Consecutive



CV types…

Consecutive

```
> cvf <- cvFolds( nrow(data), K=NUM_FOLDS, type="consecutive" )
> cvf$which
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [49] 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4
 [97] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
[145] 5 5 5 5 5 5
```

Interleaved

```
> cvf <- cvFolds( nrow(data), K=NUM_FOLDS, type="interleaved" )
> cvf$which
  [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
 [49] 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3
 [97] 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1
[145] 5 1 2 3 4 5
```

Each with K=5 "folds"

CV types…

`library(cvTools)`

Consecutive

```
> cvf <- cvFolds( nrow(data), K=NUM_FOLDS, type="consecutive" )
> cvf$which
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [49] 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4
 [97] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
[145] 5 5 5 5 5 5
```

Interleaved

```
> cvf <- cvFolds( nrow(data), K=NUM_FOLDS, type="interleaved" )
> cvf$which
  [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3
 [49] 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1
 [97] 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4
[145] 5 1 2 3 4 5
```

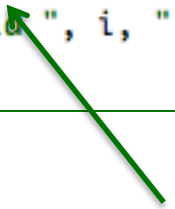
Each with K=5 "folds"
For a dataset with nrow=150 observations

CV in R...

`library(cvTools)`

Consecutive

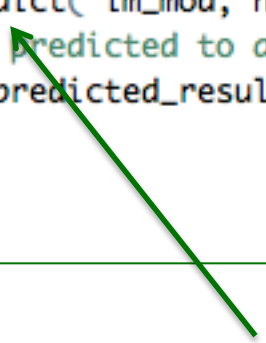
```
# cross-validation error
NUM_FOLDS <- 5
cvf <- cvFolds( nrow(data), K=NUM_FOLDS, type="consecutive" )
w <- cvf$which

# run the cross-validation
for (i in 1:NUM_FOLDS)
{
  data.train <- data[w!=i,]  # training data
  data.notrain <- data[w==i,]   # non-training data
  data.test <- data.notrain[,-3]   # removes the predicted column
  # not really necessary, but safe...
  actual_results <- data.notrain$Petal.Length
  error <- model_and_check( data.train, data.test, actual_results )
  cat("error at fold ", i, " is ", error, "\n")
}
```



can handle ***any*** model you'd like...

CV in R...

```r
# builds a model and checks the results vs. actual results...
model_and_check <- function( data.train, data.test, actual_results )
{
  # build the model
  lm_mod <- lm(Petal.Length ~ ., data=data.train )
  # get the predicted results
  predicted_results <- predict( lm_mod, newdata=data.test )
  # compute the error from predicted to actual
  error <- compute_error( predicted_results, actual_results, type="rmse" )
  # return the error
  return(error)
}
```



can handle ***any*** prediction function you'd like...

NNets, for example, use **`compute`**

CV in R...

RMSE

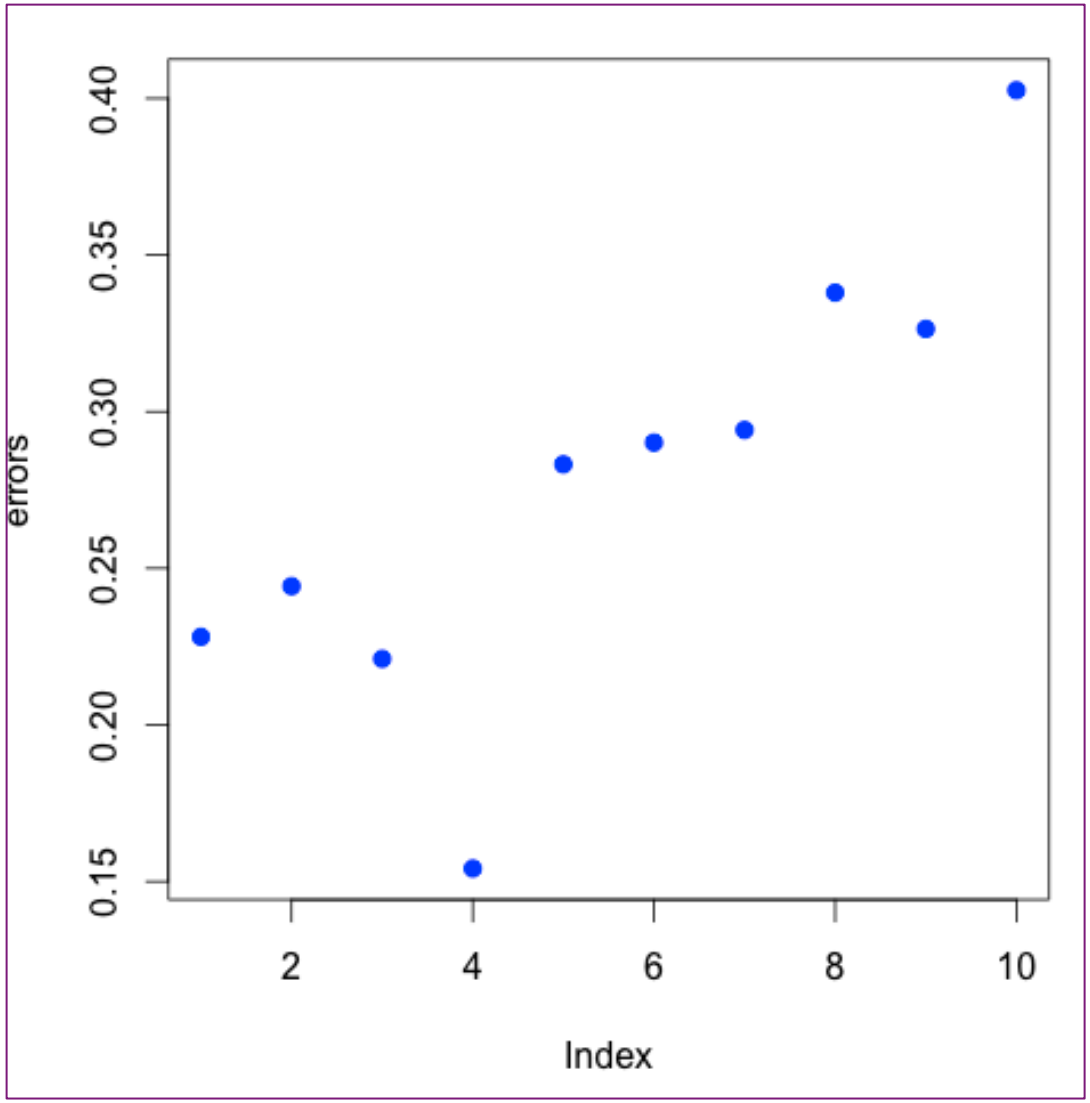
MAE

RTMSE

```r
# computes the error rate for continuous values
compute_error <- function( predicted_results, actual_results, type="rmse",
{
  # errors from cvTools / cost
  # mspe: mean squared prediction error
  # rmspe: root...
  # mape: mean absolute prediction error
  # tmspe: trimmed mean squared prediction error, trim=0.25 by default
  # rtmspe: root...
  spe <- (predicted_results - actual_results)^2
  ape <- abs(predicted_results - actual_results)
  mape <- mean(ape) # mean abs prediction error
  mspe <- mean(spe)

  rmspe <- sqrt(mspe) # root mean squared pred. error
  if (type=="rmse" || type=="rmspe") { return(rmspe) } # usual RMSE
  if (type=="rtmse" || type=="rtmspe") {                # trimmed error
    trim_index <- ceiling(length(spe)*(1-trim_frac))
    tspe <- sort(spe)[1:trim_index]
    tmspe <- mean(tspe)
    rtmspe <- sqrt(tmspe) # trimmed!
    return(rtmspe)
  }
  if (type=="mape" || type=="mae") { return(mape) } # abs value
  # anything else, return rmspe
  return(rmspe)
}
```

can handle ***any*** error you'd care to define...

CV in R…

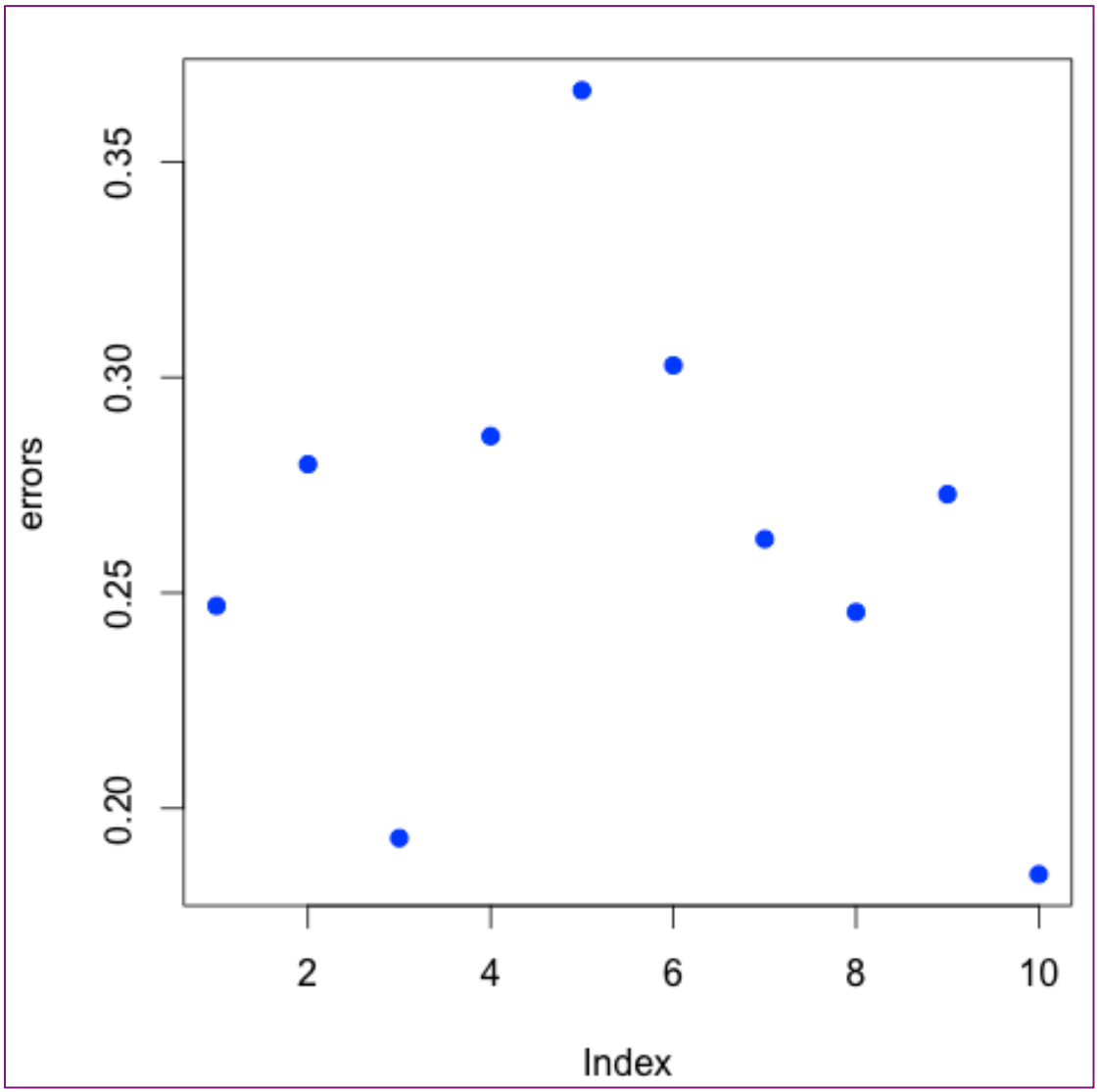


iris data

10-fold cross-validation for iris data… thoughts?

CV in R...

iris data

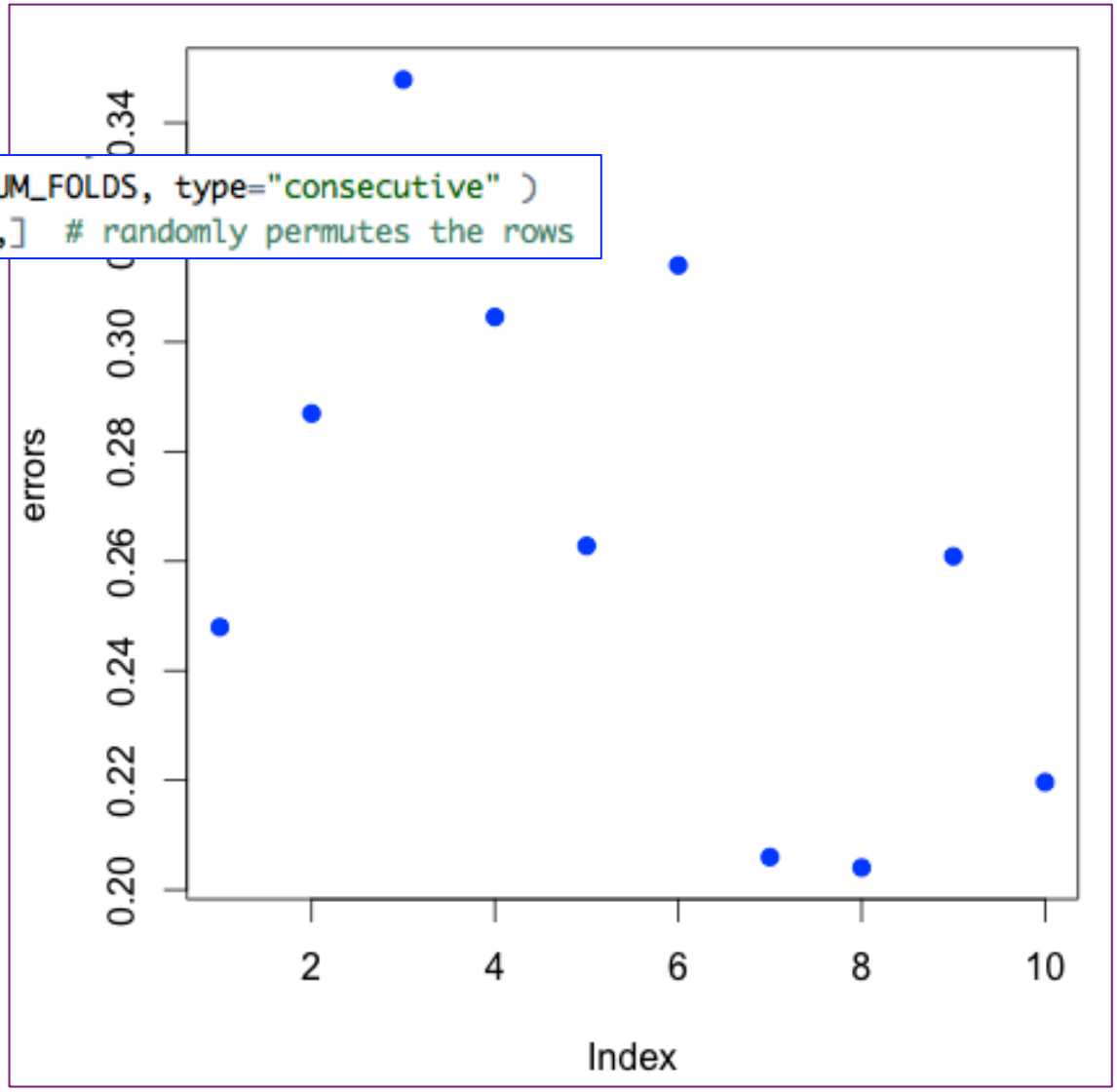


10-fold ***interleaved*** cross-validation

CV in R...

```
cvf <- cvFolds( nrow(data), K=NUM_FOLDS, type="consecutive" )
data <- data[sample(nrow(data)),]  # randomly permutes the rows
```

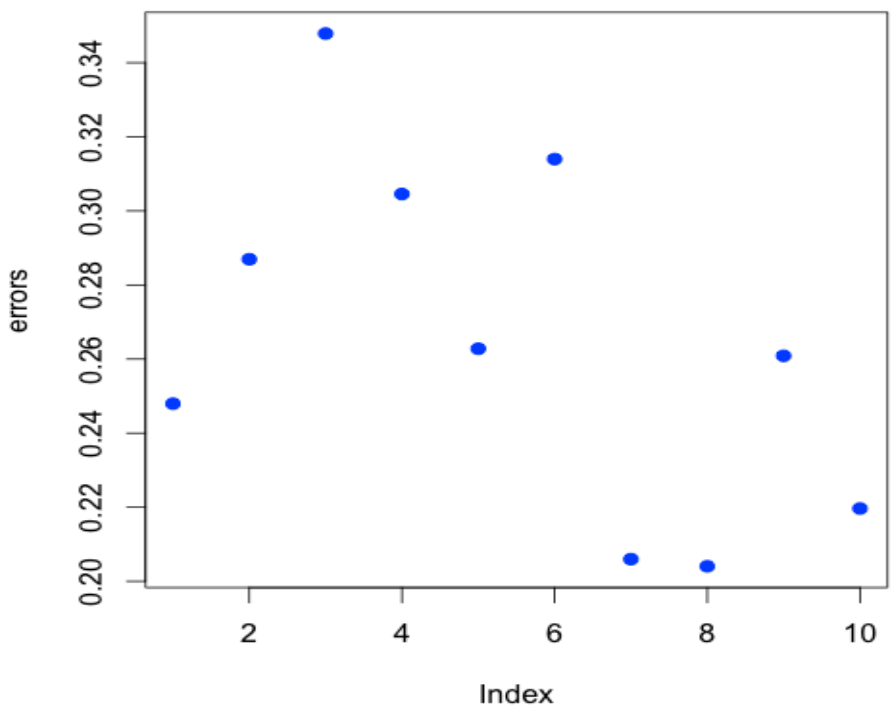
sample(1:4)

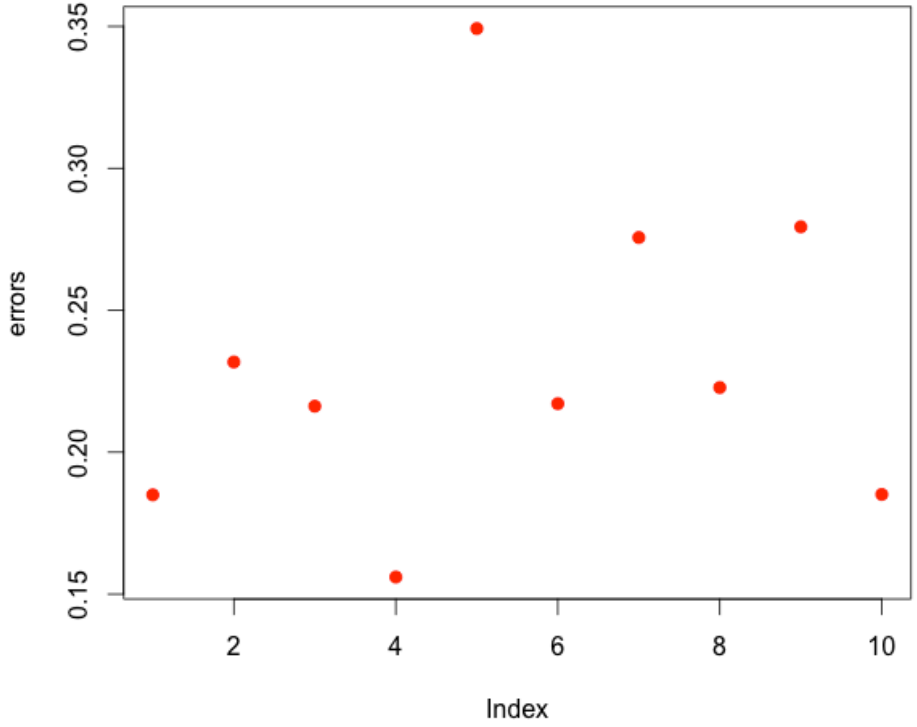


add randomness using sample

Model selection…

comparing linear model vs. NN





Lin. ~ average RMSE = 0.272

NN ~ average RMSE = 0.231

and the NN didn't use the species!

Model selection…

need a different function for NNets

```
# here is a model_and_check function for NNets
model_and_check_nn <- function( data.train, data.test, actual_results )
{
  #library("neuralnet")
  # build the model
  model <- neuralnet(Petal.Length ~
                       Sepal.Length+Sepal.Width+Petal.Width,
                     data=data.train, lifesign="full", # minimal or none
                     threshold = 0.08,
                     hidden=c(5))  # 3 neurons in the hidden layer
  plot.new()  # optional!
  plot(nn)
  # get the predicted results here, with compute
  full_predicted_results <- compute( nn, data.test )
  predicted_results <- full_predicted_results$net.result
  # compute the error from predicted to actual
  error <- compute_error( predicted_results, actual_results, type="rmse" )
  # return the error
  return(error)
}
```

special "predict" function

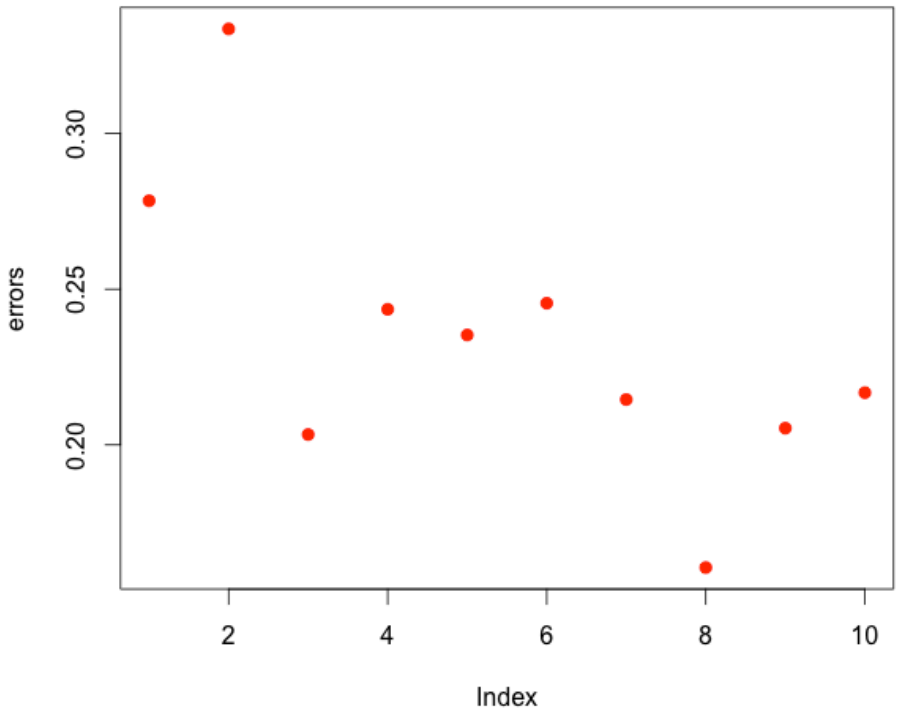
parameters

the NN didn't use the species!

but, how do we know what parameters to choose?

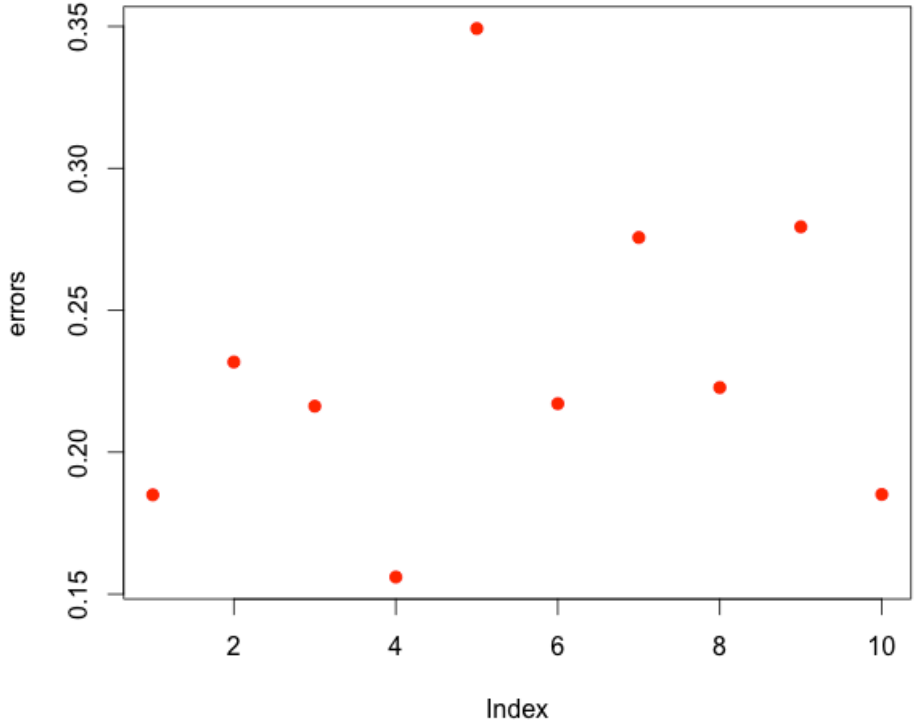
Model selection…

Even more common for ***parameter tuning***…



NN ~ average RMSE = 0.233

6 hidden neurons ~ 2 layers

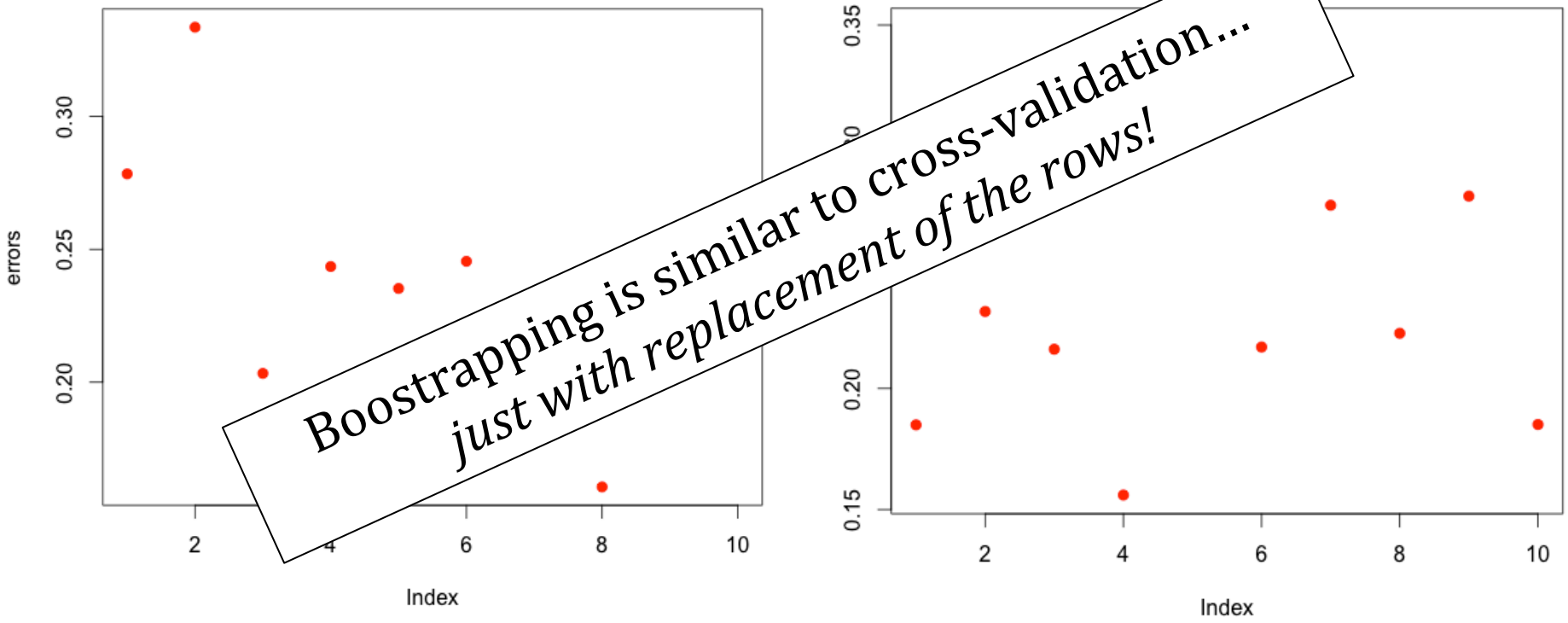


NN ~ average RMSE = 0.231

5 hidden neurons ~ 1 layer

Model selection…

Even more common for ***parameter tuning***…



NN ~ average RMSE = 0.233

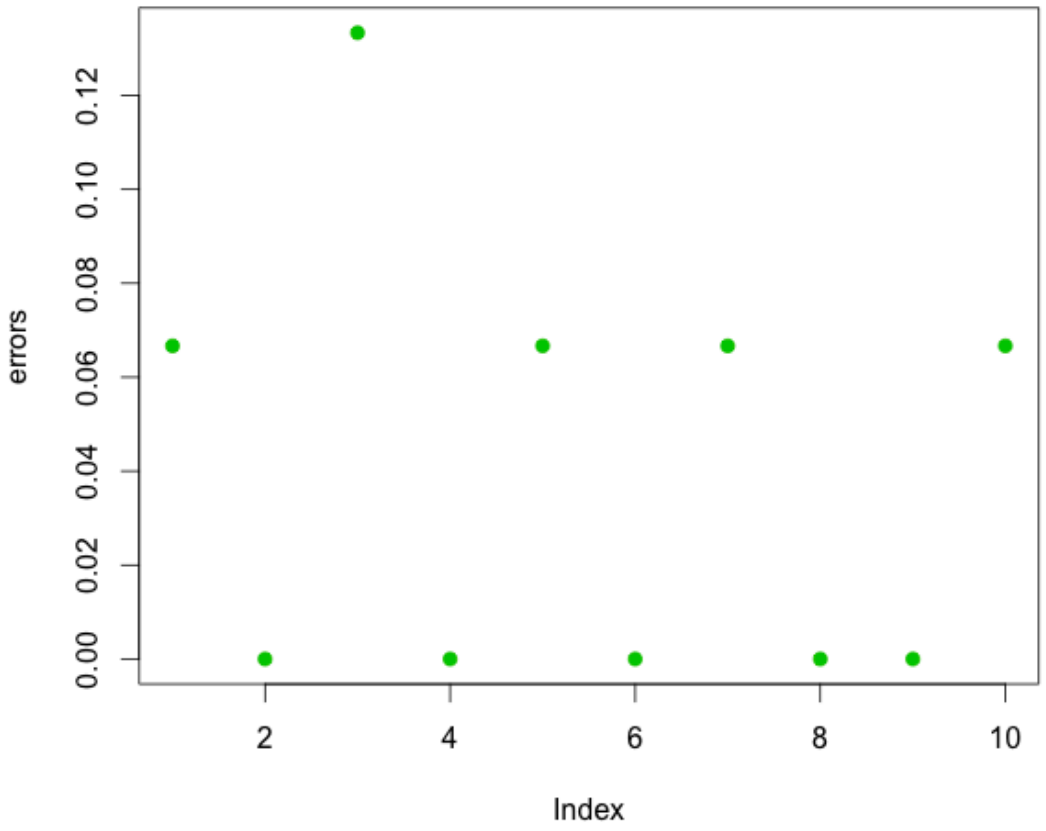
6 hidden neurons ~ 2 layers

NN ~ average RMSE = 0.231

5 hidden neurons ~ 1 layer

Model selection…

For ***classification***, you can measure accuracy rates…



knn ~ average accuracy = 96%

or you can weight false positives and false negatives differently…

Gradient-boosted trees…

one more model worth mentioning (at least!)…

as good (or better) than random forests

similar – in spirit – to random forests

… *but also provides a measure of importance of the variables involved*

fights!

Names [edit]

The method goes by a wide variety of names. Title of the original publication[1] refers to it as a "Gradient Boosting Machine" (GBM). That same publication and a later one[2] by J. Friedman also use the names "Gradient Boost", "Stochastic Gradient Boosting" (emphasizing the random subsampling technique), "Gradient Tree Boosting" and "TreeBoost" (for specialization of the method to the case of decision trees as base learners.)

A popular open-source implementation[6] calls it "Generalized Boosting Model". Sometimes the method is referred to as "functional gradient boosting", "Gradient Boosted Models" and its tree version is also called "Gradient Boosted Decision Trees" (GBDT) or "Gradient Boosted Regression Trees" (GBRT). Commercial implementations from Salford Systems use the names "Multiple Additive Regression Trees" (MART) and TreeNet, both trademarked.

You're successful when everyone claims you've contributed almost nothing at all…!

Gradient-boosted trees…

one more model worth mentioning (at least!)…

as good (or better) than random forests

similar – in spirit

also provides a measure of importance of the variables involved

	Algorithm 1: Gradient_TreeBoost
1	$F_0(\mathbf{x}) = \arg\min_\gamma \sum_{i=1}^{N} \Psi(y_i, \gamma)$
2	For $m = 1$ to M do:
3	$\bar{y}_{im} = -\left[\frac{\partial \Psi(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, \; i = 1, N$
4	$\{R_{lm}\}_1^L = L - \text{terminal node } tree(\{\bar{y}_{im}, \mathbf{x}_i\}_1^N)$
5	$\gamma_{lm} = \arg\min_\gamma \sum_{\mathbf{x}_i \in R_{lm}} \Psi(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$
6	$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu \cdot \gamma_{lm} 1(\mathbf{x} \in R_{lm})$
7	endFor

Gradient-boosted trees…

proposed by Friedman (Stanford) ~ 2000

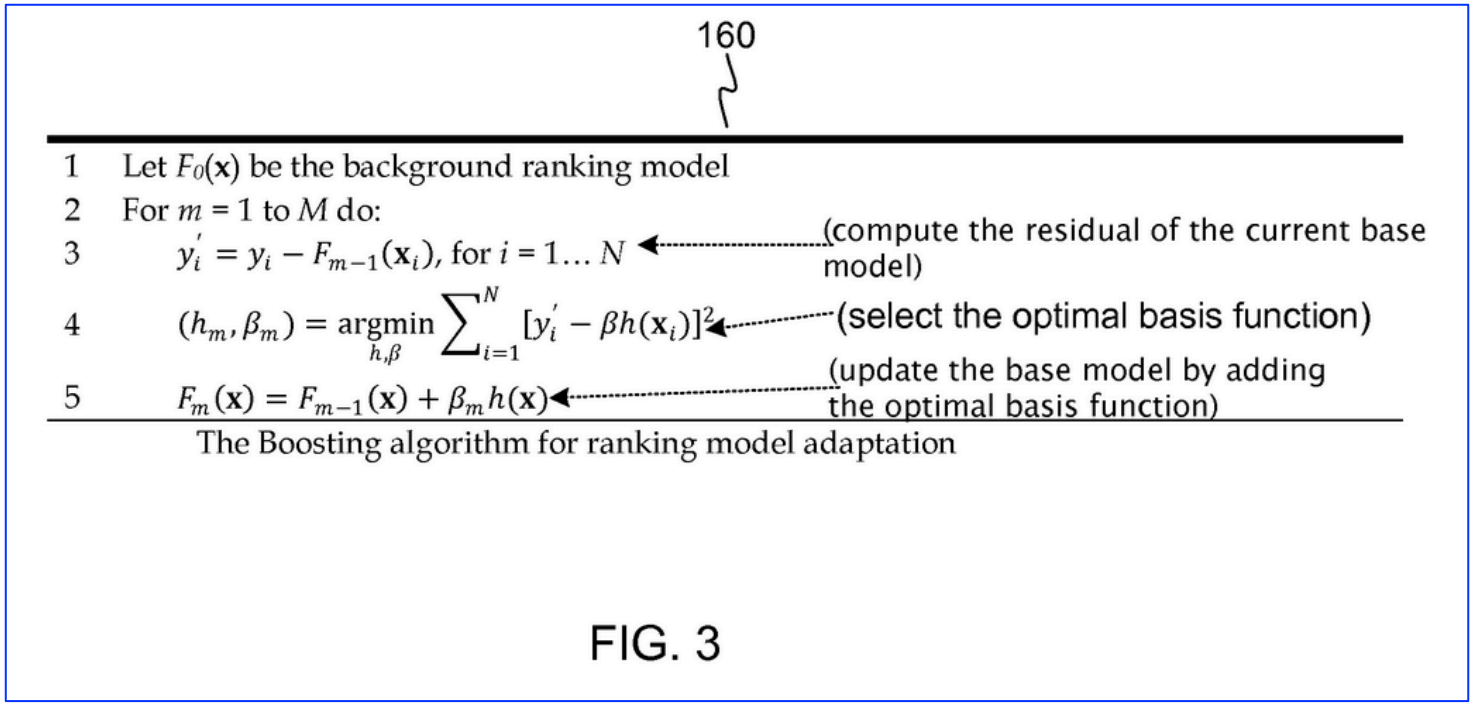
Aargh!

Algorithm 1: Gradient_TreeBoost

1. $F_0(\mathbf{x}) = \arg\min_\gamma \sum_{i=1}^{N} \Psi(y_i, \gamma)$
2. For $m = 1$ to M do:
3. $\quad \tilde{y}_{im} = -\left[\frac{\partial \Psi(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, \; i = 1, N$
4. $\quad \{R_{lm}\}_1^L = L - \text{terminal node } tree(\{\tilde{y}_{im}, \mathbf{x}_i\}_1^N)$
5. $\quad \gamma_{lm} = \arg\min_\gamma \sum_{\mathbf{x}_i \in R_{lm}} \Psi(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$
6. $\quad F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu \cdot \gamma_{lm} 1(\mathbf{x} \in R_{lm})$
7. endFor

Gradient-boosted trees…

We'll use the version from the patent filing…



The Boosting algorithm for ranking model adaptation

FIG. 3

Gradient-boosted trees…

We'll use the version from the patent filing…

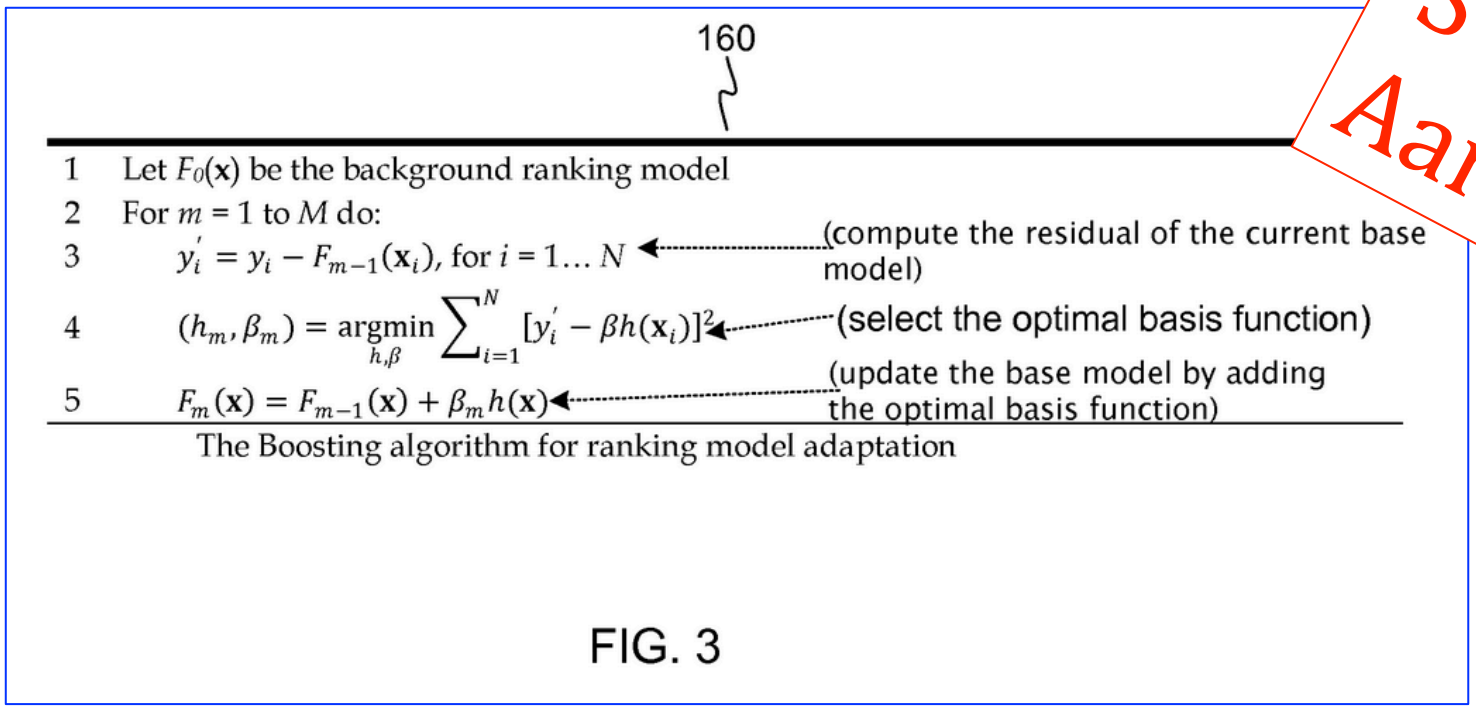
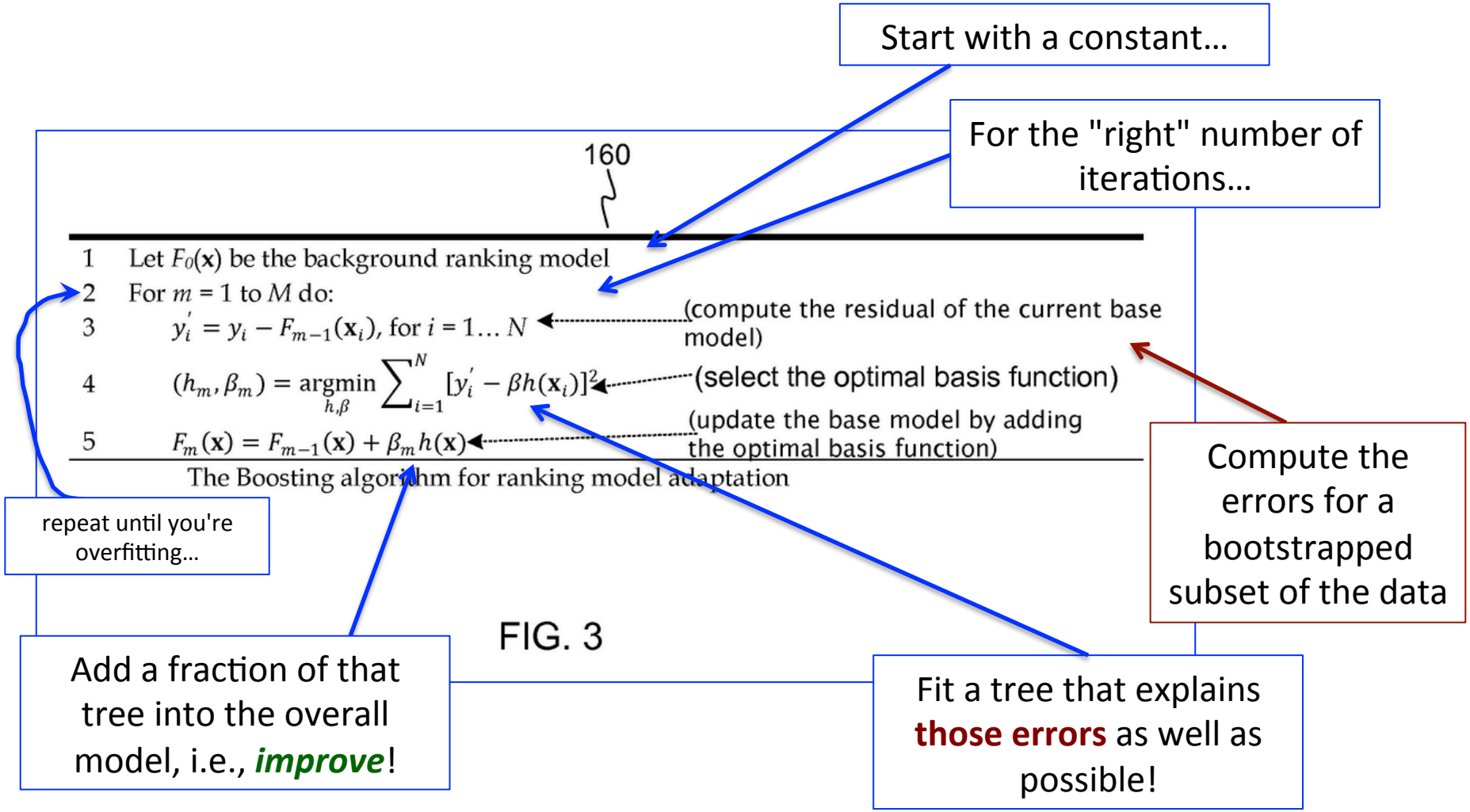


FIG. 3

STILL Aargh!

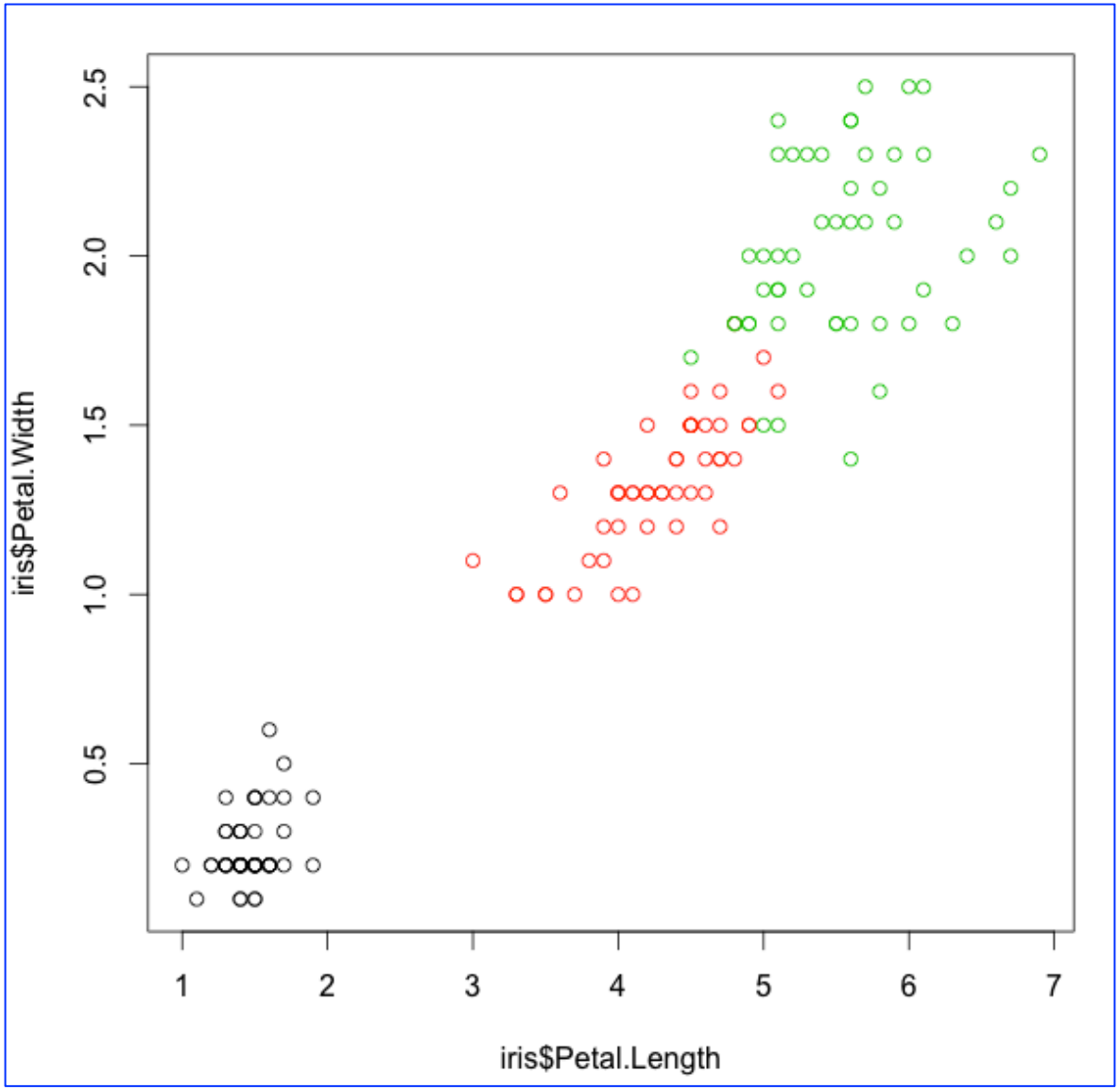
Gradient-boosted trees…

We'll use the version from the patent filing…

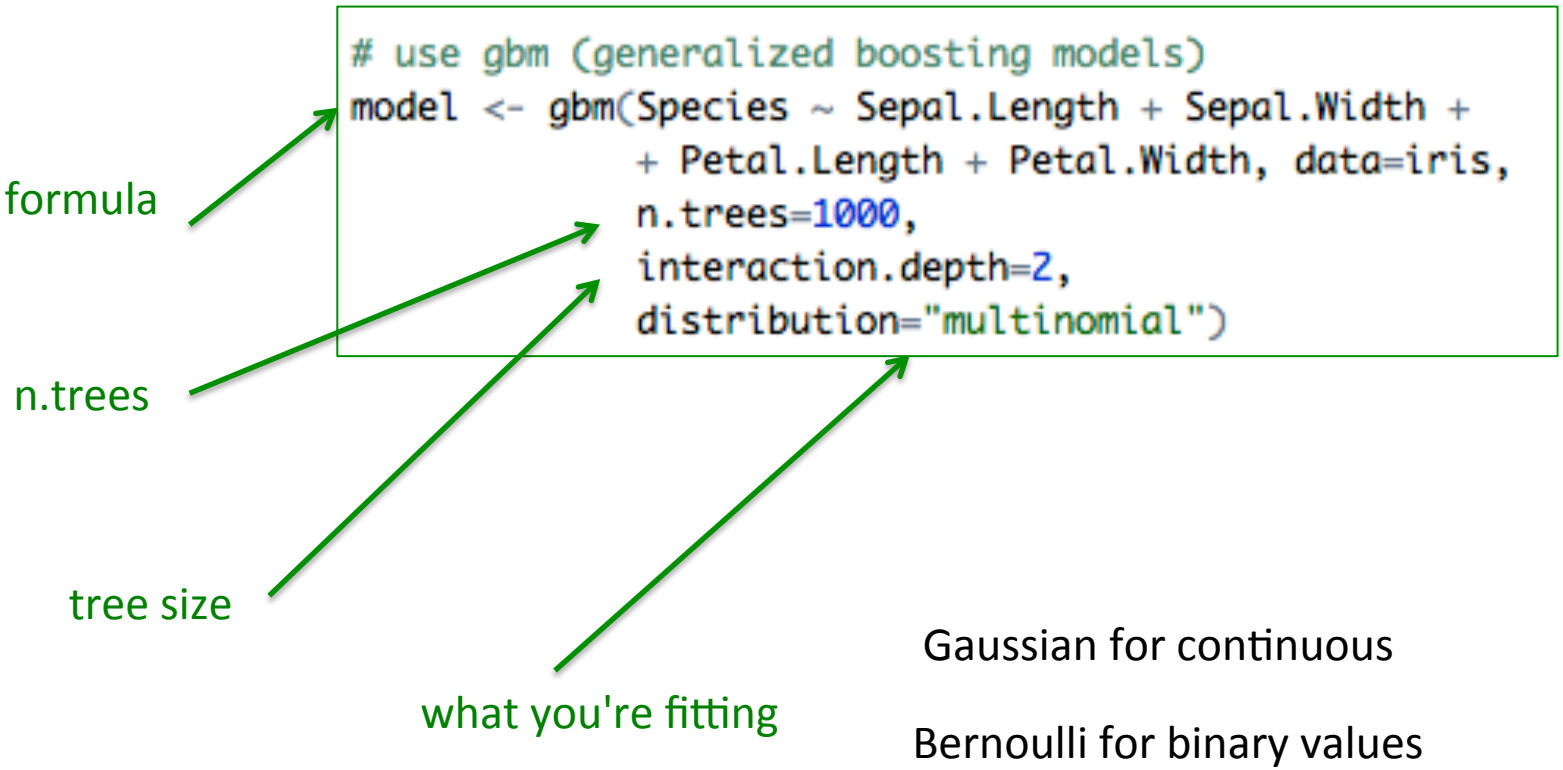


Gradient-boosted trees…

in practice…



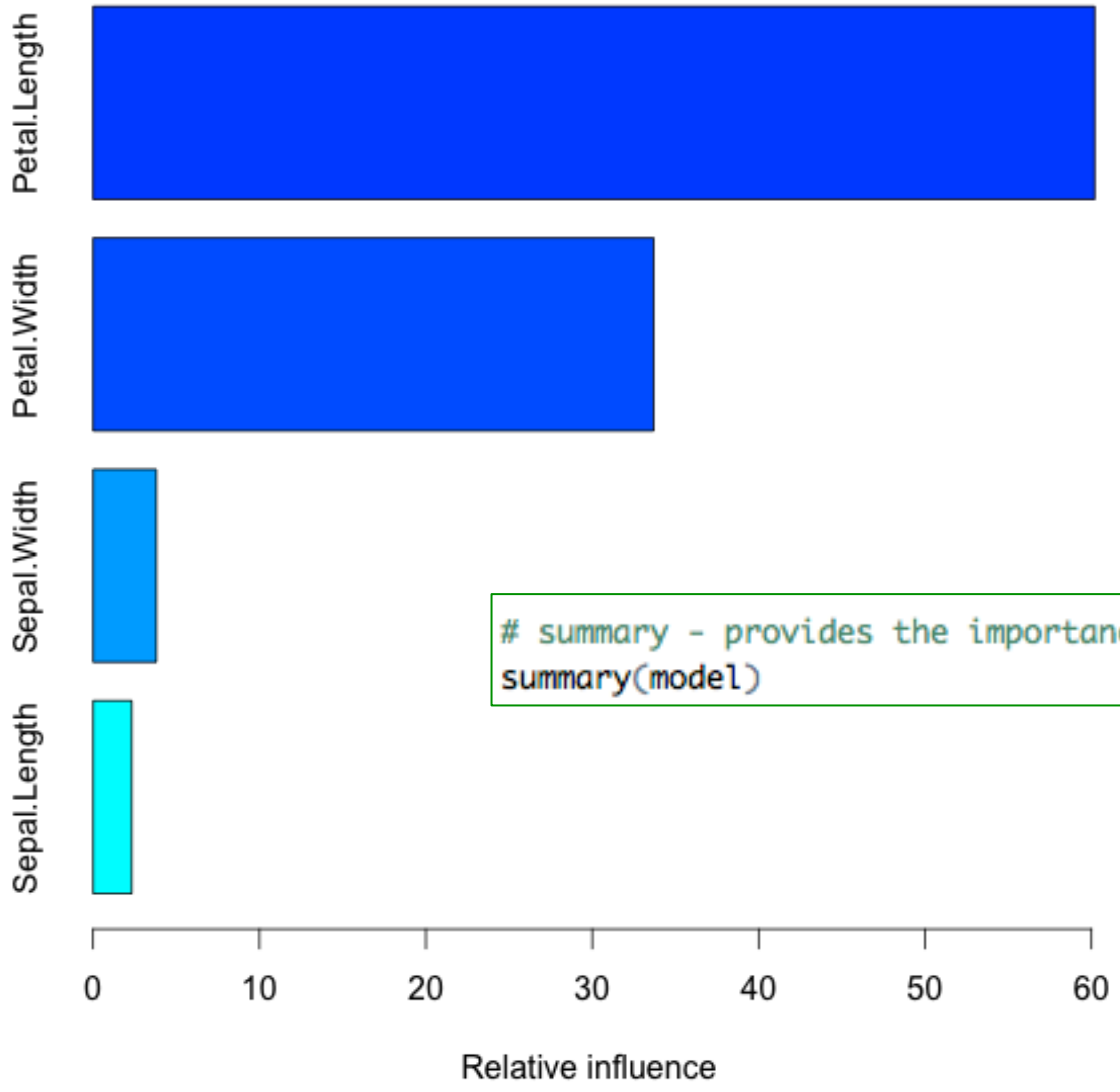
Gradient-boosted trees…



Gaussian for continuous

Bernoulli for binary values

Gradient-boosted trees…




```
# summary - provides the importance of each factor...
summary(model)
```

Predicting...

```
> # predict!
> prediction <- predict.gbm(model, newdata=iris2[45:55,],
+                           type="response",
+                           n.trees=1000)
> prediction
 [1] 0.1254753 0.1369078 0.1254753 0.1261382 0.1257360 0.1254753 0.6849997
 [8] 0.6849997 0.5395748 0.7350768 0.7219110
```

```
> iris2[45:55,]
   Sepal.Length Sepal.Width Petal.Length Petal.Width    Species i
45          5.1         3.8          1.9         0.4     setosa
46          4.8         3.0          1.4         0.3     setosa
47          5.1         3.8          1.6         0.2     setosa
48          4.6         3.2          1.4         0.2     setosa
49          5.3         3.7          1.5         0.2     setosa
50          5.0         3.3          1.4         0.2     setosa
51          7.0         3.2          4.7         1.4 versicolor
52          6.4         3.2          4.5         1.5 versicolor
53          6.9         3.1          4.9         1.5 versicolor
54          5.5         2.3          4.0         1.3 versicolor
55          6.5         2.8          4.6         1.5 versicolor
```

Other examples…

```
# this shows off gbm's ability to "tease out" important factors...

set.seed(1234)
N <- 1000
X1 <- runif(N)
X2 <- 2*runif(N)
X3 <- ordered(sample(letters[1:4],N,replace=TRUE),levels=letters[4:1])
X4 <- factor(sample(letters[1:6],N,replace=TRUE))
X5 <- factor(sample(letters[1:3],N,replace=TRUE))
X6 <- 3*runif(N)
mu <- c(-1,0,1,2)[as.numeric(X3)]

SNR <- 10 # signal-to-noise ratio
Y <- .5*(X1**2) + 2 * (X2**.5) + mu

# NOTICE: it depends on X1, X2, and X3

sigma <- sqrt(var(Y)/SNR)
Y <- Y + rnorm(N,0,sigma)

# introduce some missing values
X1[sample(1:N,size=500)] <- NA
X4[sample(1:N,size=300)] <- NA

data <- data.frame(Y=Y,X1=X1,X2=X2,X3=X3,X4=X4,X5=X5,X6=X6)
```

Model comparisons…

Characteristic	Neural nets				
Natural handling of data of "mixed" type	🔴	🔴	🟢	🟢	🔴
Handling of missing values	🔴	🔴	🟢	🟢	🟢
Robustness to outliers in input space	🔴	🔴	🟢	🔴	🟢
Insensitive to monotone transformations of inputs	🔴	🔴	🟢	🔴	🔴
Computational scalability (large N)	🔴	🔴	🟢	🟢	🔴
Ability to deal with irrelevant inputs	🔴	🔴	🟢	🟢	🔴
Ability to extract linear combinations of features	🟢	🟢	🔴	🔴	🟡
Interpretability	🔴	🔴	🟡	🟢	🔴
Predictive power	🟢	🟢	🔴	🟡	🟢

Where would you put
- k-nearest neighbors
- support vector machines
- decision trees
- gradient boosted trees

🟢 excellent

🟡 moderate

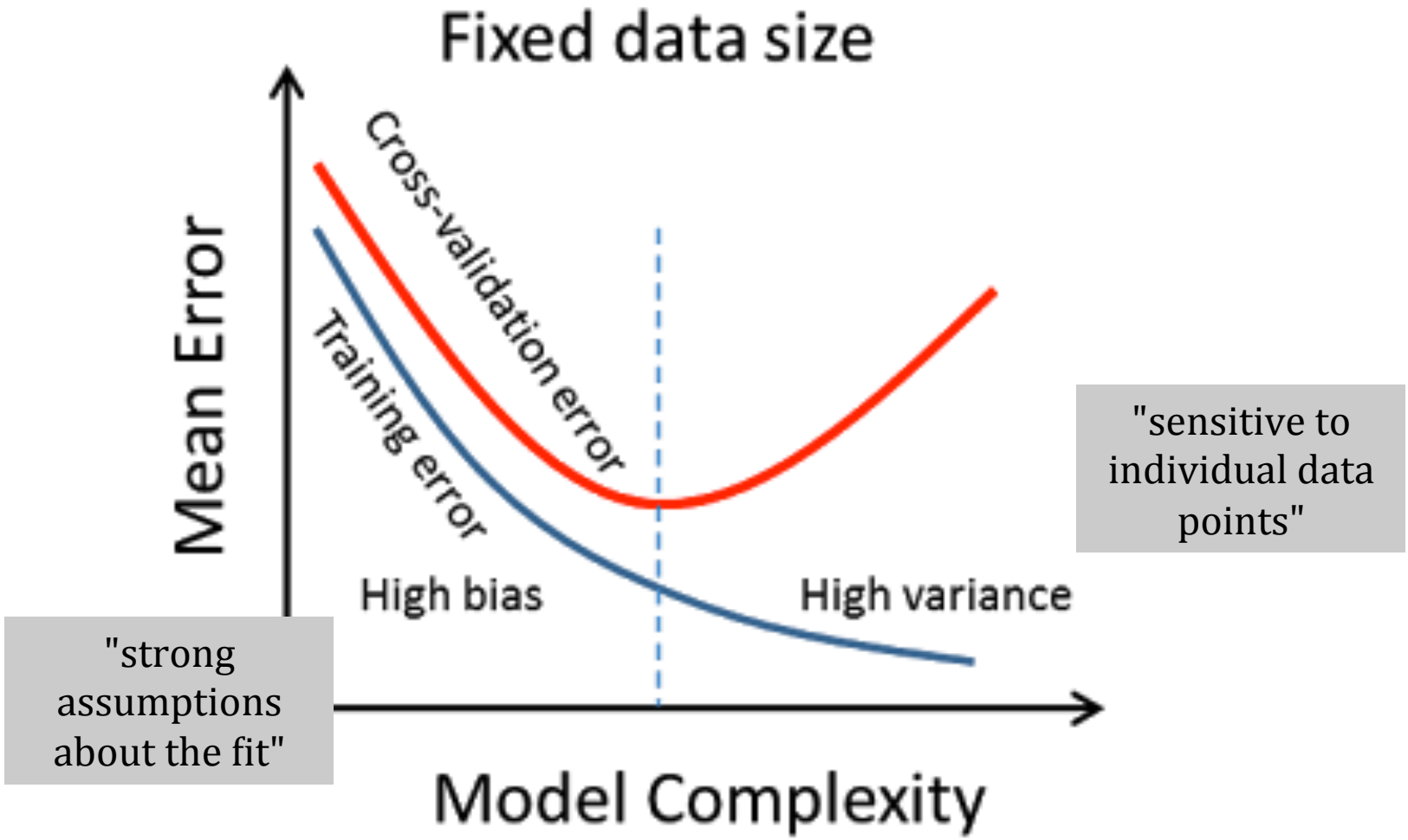
🔴 poor perf.

Model comparisons…

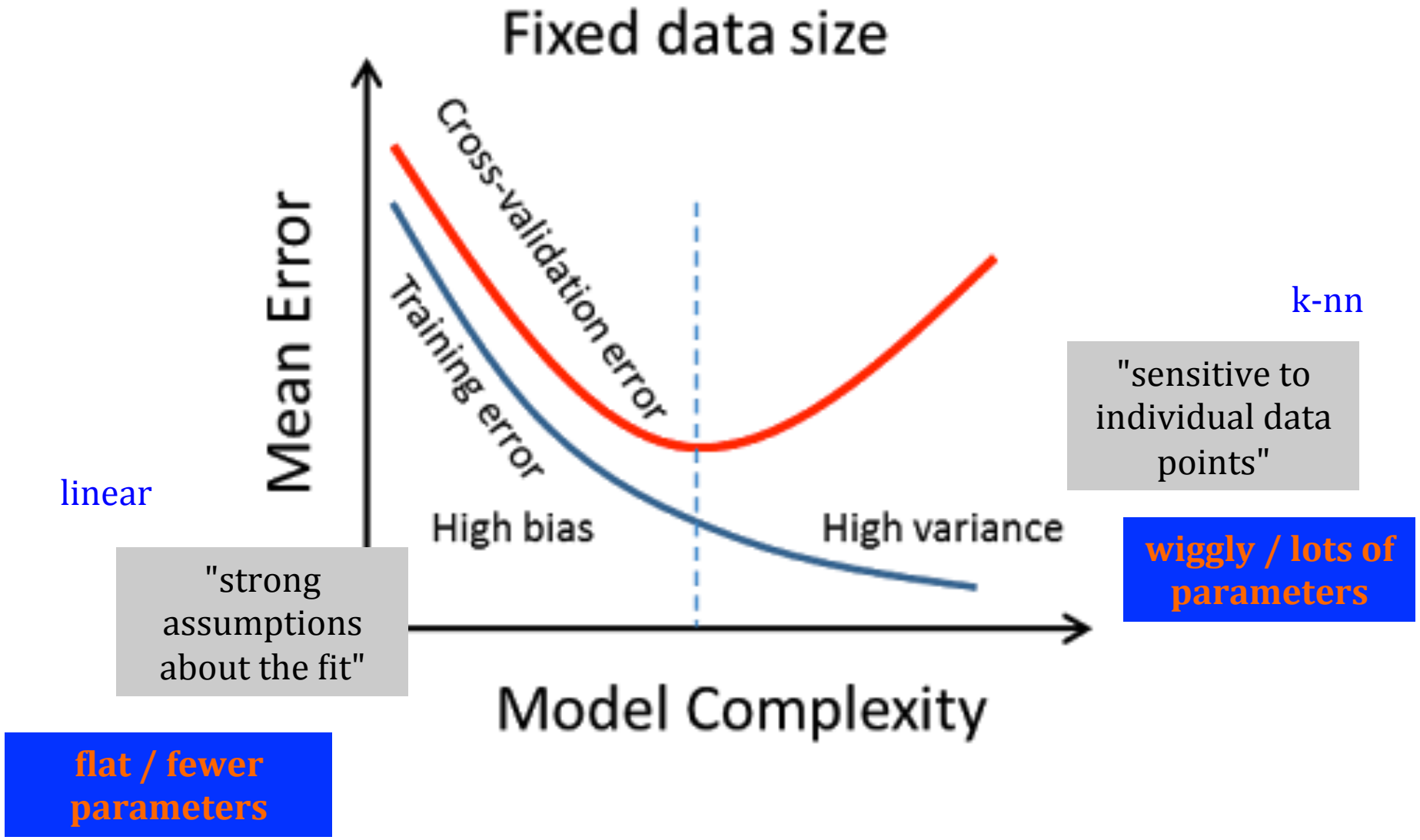
gradient boosted trees

Characteristic	Neural nets	SVM	Trees	MARS	k-NN, kernels
Natural handling of data of "mixed" type	🔴	🔴	🟢	🟢	🔴
Handling of missing values	🔴	🔴	🟢	🟢	🟢
Robustness to outliers in input space	🔴	🔴	🟢	🔴	🟢
Insensitive to monotone transformations of inputs	🔴	🔴	🟢	🔴	🔴
Computational scalability (large N)	🔴	🔴	🟢	🟢	🔴
Ability to deal with irrelevant inputs	🔴	🔴	🟢	🟢	🔴
Ability to extract linear combinations of features	🟢	🟢	🔴	🔴	🟡
Interpretability	🔴	🔴	🟡	🟢	🔴
Predictive power	🟢	🟢	🔴	🟡	🟢

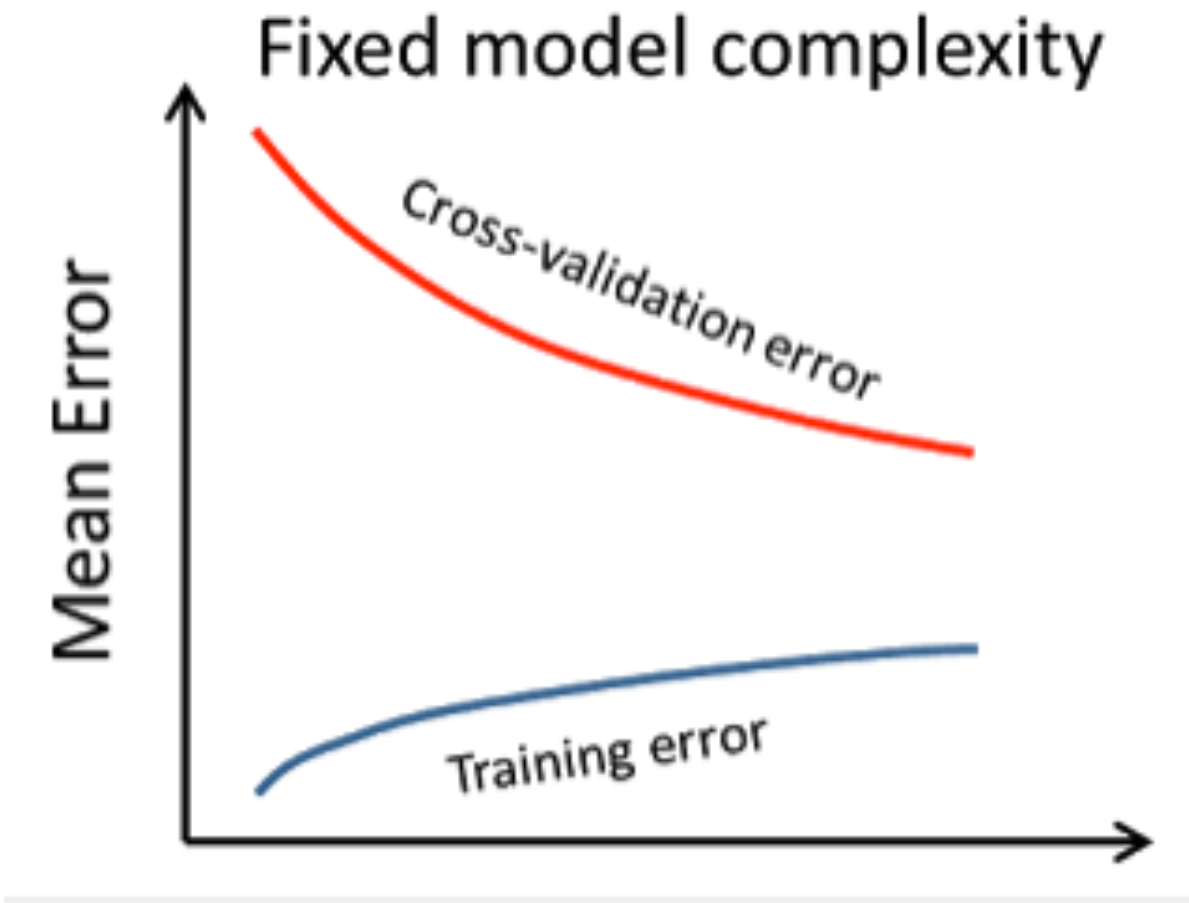
Model comparisons…



Model comparisons…

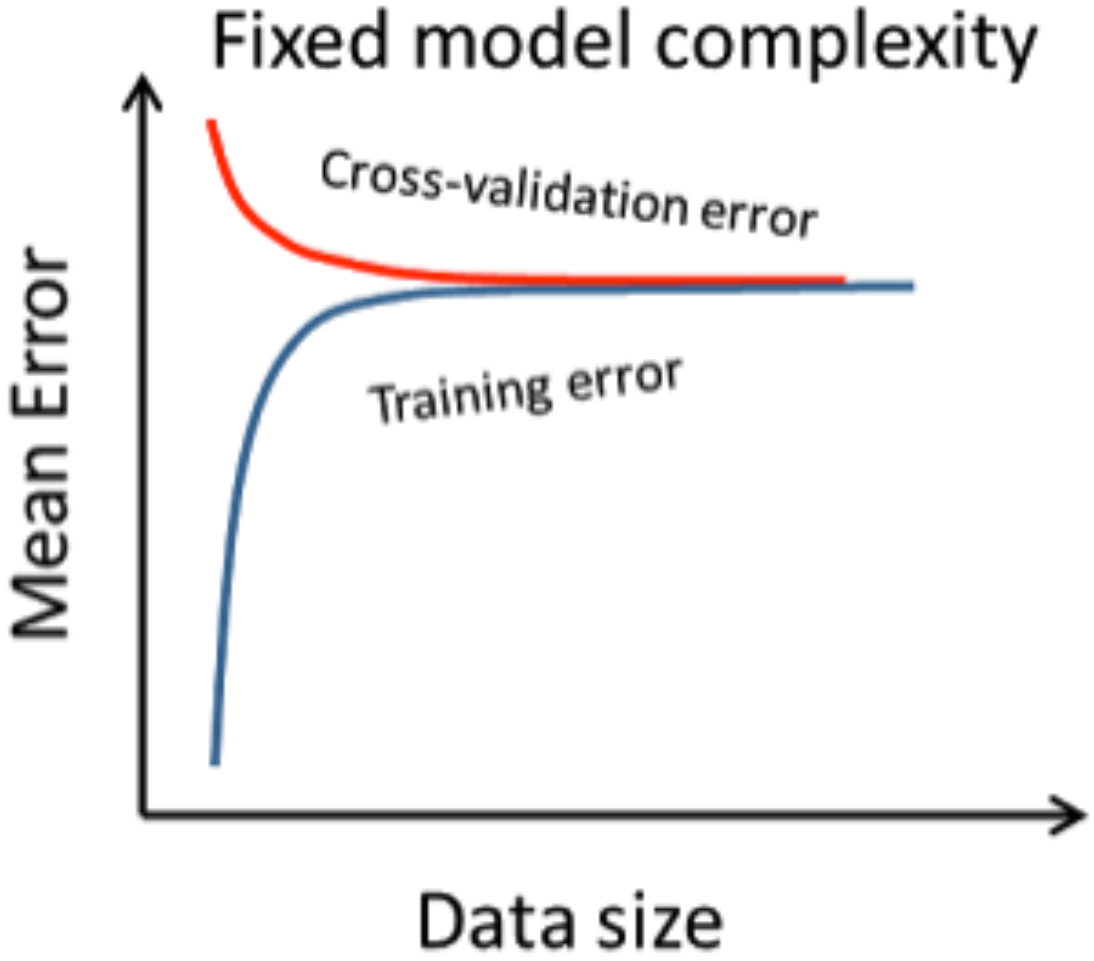


Now choose a model…



More data will improve things!

Now choose a model…



More data will **NOT** help!

features!

- Add more input features by talking to domain experts to identify more input signals and collect them.
- Add more input features by combining existing input features in a non-linear way.

Project schedule

Good luck ~ let's chat!

dataset

- Explain the dataset
- Initial set of questions about it
- A desired predictive model
- Two initial visualizations (in R) (read the data!)
- Run an initial SVM, NN, and k-NN classifier
- Other systems OK, but only *in addition*

selection and gradient-boosted classifiers

Due 4/20: feature selection and exploration

- Define features/subsets of your data
- Try the unsupervised algorithms
  hierarchical and k-means clustering
- Visualize and summarize the results

4/22: example final project talk / short session

Due 4/27: model selection / predictive modeling

- Decide on a cross-validation strategy
  10-fold, leave-one-out, parameters to test…
- Try all of the appropriate algorithms
  linear and/or logistic regression, decision trees, random forests, SVMs, NNs, and k-NNs
- Run them and summarize the results
- Ensemble vs. single-algorithm choice

On 4/29: final-project talks

Final project report due 5/11/2013 (No class 5/6/2013)

See if you can identify which is which:

Red, Green, Blue, Hue, Saturation, Value (Lightness)

Extra challenge: what colors are ***behind*** and ***under*** the can?

Project, part 1: *Data!*

Reminder: it's due this SAT, not Tue.

Model *selection*

Custom-built functions

Linear Regression
Logistic Regression

Linear Regression
Logistic Regression

Which one do I choose?

High-level considerations

Custom-built functions

Linear Regression
Logistic Regression

See tabs at HMC…

Linear Regression
Logistic Regression

Which one do I choose?

Cross-validation

Custom-built functions

Linear Regression
Logistic Regression

See tabs at HMC…

Linear Regression
Logistic Regression

Which one do I choose?

Which model?

if you trust the fit much more than the data…

FIGURE 2.1. *A classification example in two dimensions. The classes are coded as a binary variable (BLUE = 0, ORANGE = 1), and then fit by linear regression. The line is the decision boundary defined by $x^T\hat{\beta} = 0.5$. The orange shaded region denotes that part of input space classified as ORANGE, while the blue region is classified as BLUE.*

How much do you trust your data?

if you trust the data 100% (no model at all!)

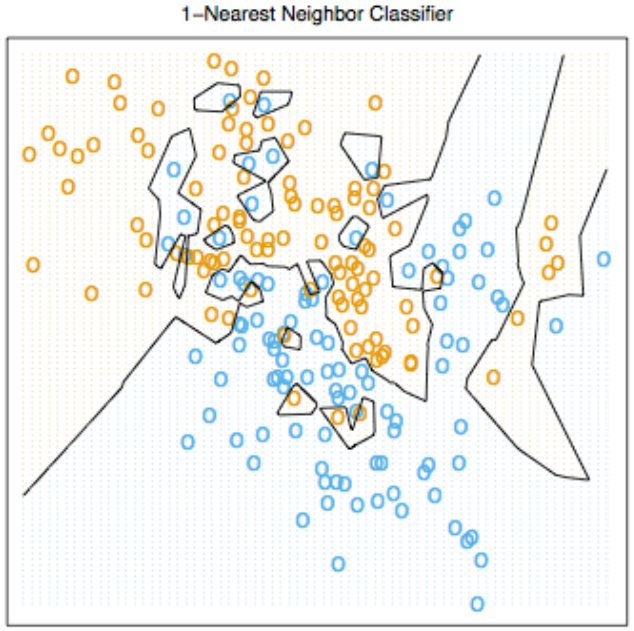
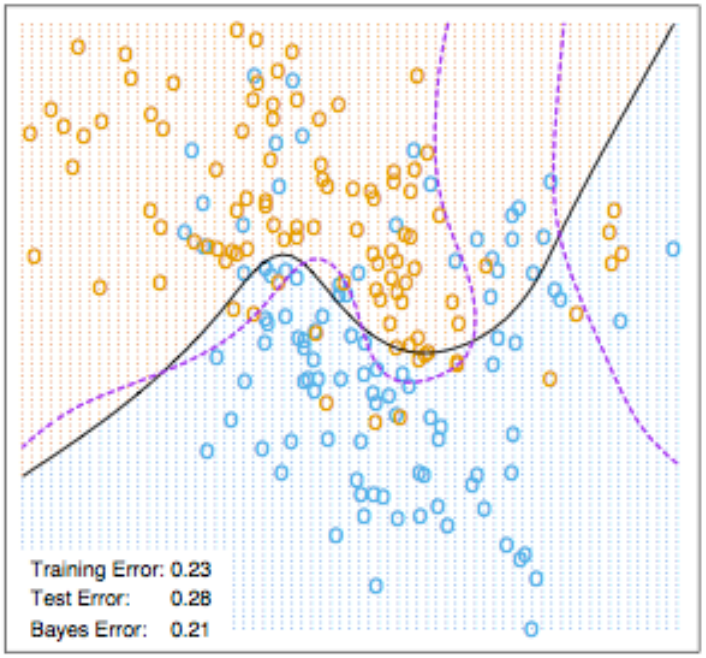


FIGURE 2.3. *The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable (BLUE = 0, ORANGE = 1), and then predicted by 1-nearest-neighbor classification.*

Which model?

How much do you trust your data?

intermediate positions…



inear rnel, ppro: and $C = 1$ worked well in both (close to Bayes optimal), as mig of Gaussians. The broken purpl boundary.

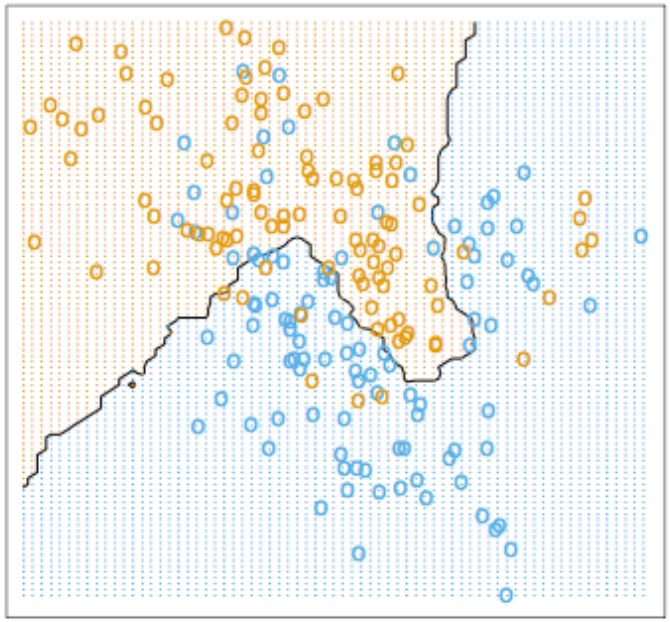


FIGURE 2.2. *The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable (BLUE = 0, ORANGE = 1) and then fit by 15-nearest-neighbor averaging as in (2.8). The predicted class is hence chosen by majority vote amongst the 15-nearest neighbors.*

Which model?

How much do you trust your data?

intermediate positions: SVM…

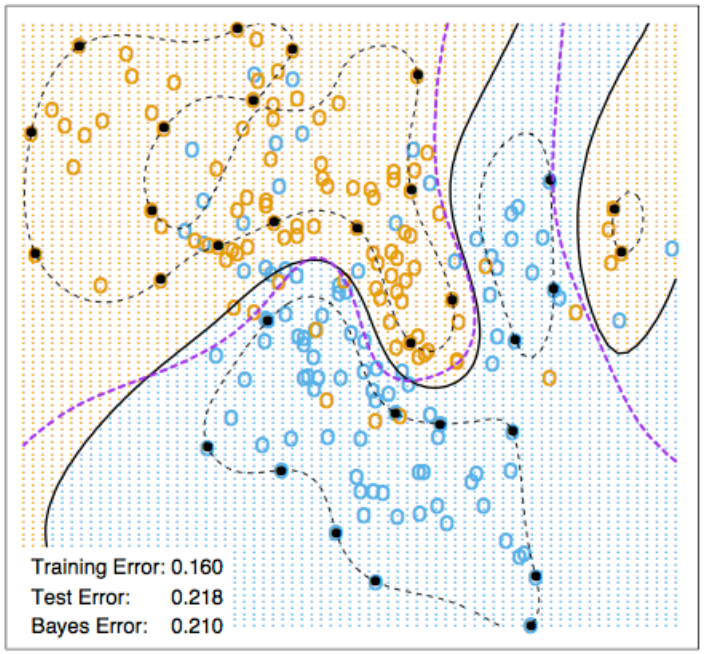


FIGURE 12.3. *Two nonlinear SVMs for the mixture data. The upper plot uses a 4th degree polynomial kernel, the lower a radial basis kernel (with $\gamma = 1$). In each case C was tuned to approximately achieve the best test error performance, and $C = 1$ worked well in both cases. The radial basis kernel performs the best (close to Bayes optimal), as might be expected given the data arise from mixtures of Gaussians. The broken purple curve in the background is the Bayes decision boundary.*

Good luck with Hw#7!

and as a reminder: it's due on Sat.