# Data Mining
# W4240 Sections 001, 003/004

Lauren A. Hannah

Columbia University, Department of Statistics

September 23, 2014

# Outline

# Administrative Notes

Homework 1 was due last week.

- turn in single pdf less than 4MB
  - (pdf pictures are large, especially color ones)
  - (png or jpg are smaller)
  - (save as black and white...)
- turn in one .R script per problem named as outline on HW
- do not compress (no RARs, TARs, ZIPs, etc)
- as per lecture 1: code checking!

**Any homework not meeting these conditions will get a 0!**

# Administrative Notes

Lecture 1 promise: code will be checked. Get 0 for copying code.

```
#----- START YOUR CODE BLOCK HERE -----#
par(mfrow=c(1,1))
for(i in 1:3){
  for(j in 1:3){
    filename=sprintf("CroppedYale/%s/%s_%s.pgm",dir_list_1[pic_list[i]],dir_lis
    a=read.pnm(file=filename)
    pic_data[[(i-1)*3+j]]=getChannels(a)
  }
}
faces_matrix_row1=cbind(pic_data[[1]],pic_data[[2]],pic_data[[3]])
faces_matrix_row2=cbind(pic_data[[4]],pic_data[[5]],pic_data[[6]])
faces_matrix_row3=cbind(pic_data[[7]],pic_data[[8]],pic_data[[9]])
faces_matrix=rbind(faces_matrix_row1,faces_matrix_row2,faces_matrix_row3)
#----- END YOUR CODE BLOCK HERE -----#
```

```
#----- START YOUR CODE BLOCK HERE -----#
par(mfrow=c(1,1))
for(i in 1:3){
  for(j in 1:3){
    filename=sprintf("CroppedYale/%s/%s_%s.pgm",dir_list_1[pic_list[i]],dir_lis
    a=read.pnm(file=filename)
    pic_data[[3*(i-1)+j]]=getChannels(a)
  }
}
faces_matrix_row1=cbind(pic_data[[1]],pic_data[[2]],pic_data[[3]])
faces_matrix_row2=cbind(pic_data[[4]],pic_data[[5]],pic_data[[6]])
faces_matrix_row3=cbind(pic_data[[7]],pic_data[[8]],pic_data[[9]])
faces_matrix=rbind(faces_matrix_row1,faces_matrix_row2,faces_matrix_row3)

#----- END YOUR CODE BLOCK HERE -----#
```

# Outline

# Supervised Learning

Recall from before...

$$
\begin{aligned}
X = (X_1, X_2, \ldots, X_p)^T & \qquad \text{inputs} \\
Y & \qquad \text{output} \\
Y = f(X) + \epsilon & \qquad \text{relationship}
\end{aligned}
$$

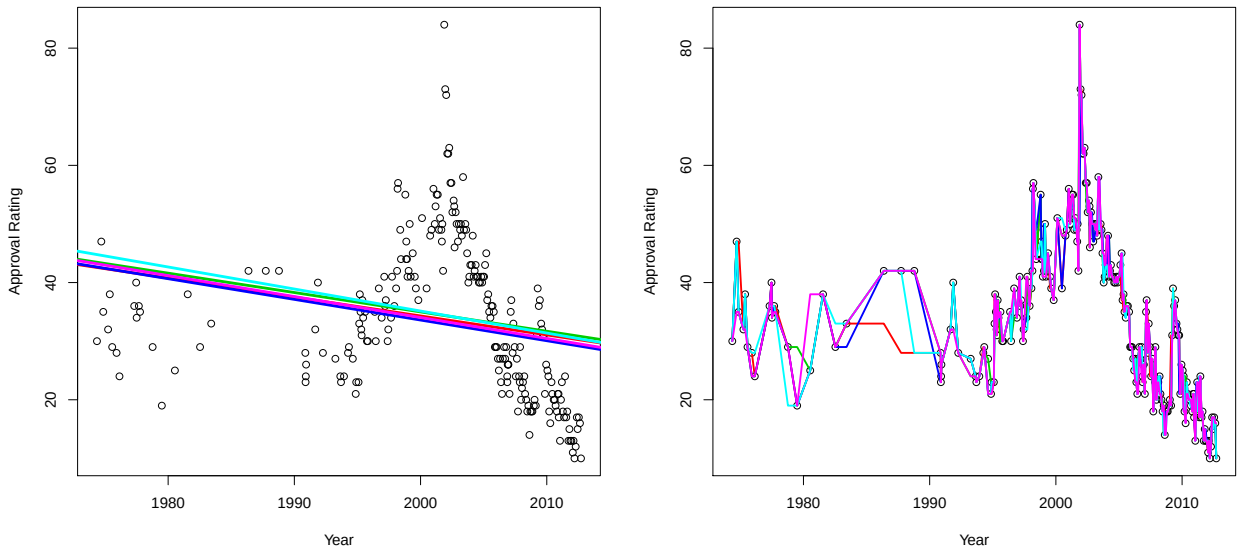
We are interested in studying $f$ in two settings: regression and classification.

**Regression:** $Y$ has continuous values, like \$81,200 or 72.

**Classification:** $Y$ has categorical values, like low/medium/high or red/green/blue

# Supervised Learning

Recall from before...




We looked at least squares linear regression and $k-$nearest neighbors.

# Linear Regression

Linear regression:

- (we haven't covered this yet, but we know how it looks...)
- fit a global model, restricted to be a linear function of the covariates
- Our data is $y = f(\mathbf{x}) + \epsilon$
- We fit

$$\hat{y} = \hat{f}(\mathbf{x}) = w_0 + \sum_{j=1}^{p} w_j x_j$$

- aka

$$\hat{y} = \mathbf{w}^\top \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$$

- (The function is a linear combination of the covariates)
- (And how does this look in data matrix form?)

# $k-$Nearest Neighbors Regression

Idea: fit a local model by averaging the values of the $k$ closest observations

$$\hat{y} = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$

where $N_k(x)$ is the set of observations with the $k$ smallest distances to the query point $x$

(remember also we can do $k-$nearest neighbor *classification*)

Questions:

- Is linear regression or $k$NN better?
- Which $k$ is best?
- Is there a best?

# Supervised Learning

Why is there no single best estimator?

# Supervised Learning

How do I measure an estimator? Prediction error is a logical choice:

Ideally: **Expected Mean Squared Error**:

$$\mathbb{E}\left[(Y - \hat{Y})^2 \,|\, X = x_0\right]$$

This measures the expected ability of an estimator (given random training data) to predict a new outcome at the point $x_0$. You can view this as the *generalization error*.

Why do I want this?

# Supervised Learning

Why squared error? First, assume we can parameterize our estimator $f$ by some set of parameters $\theta$.

Assume $\mathbb{P}(Y \mid X, \theta) = N(f_\theta(X), \sigma^2)$

$$\ell(\theta) = -\frac{n}{2}\log(2\pi) - n\log\sigma - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - f_\theta(x_i))^2$$

$$\propto -\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - f_\theta(x_i))^2$$

*Gaussian errors mean squared loss function!*

# Supervised Learning

Really want **Expected Mean Squared Error**:

$$\mathbb{E}\left[(Y - \hat{Y})^2 \,|\, X = x_0\right]$$

This requires:

- ability to compute expectation over all possible outcomes for $\hat{Y}$ (different training data)
- ability to compute expectation over $Y$ (know distribution of response for a given $x_0$)

I have: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$

# Outline

# Mean Squared Error

Well, what if I just approximate

$$\mathbb{E}\left[(Y - \hat{Y})^2 \,|\, X = x_0\right]$$

with my data?

$$MSE_{training} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{f}(x_i)\right)^2$$

# Mean Squared Error

New metric:

$$MSE_{training} = \frac{1}{n} \sum_{i=1}^{n} \left( y_i - \hat{f}(x_i) \right)^2$$

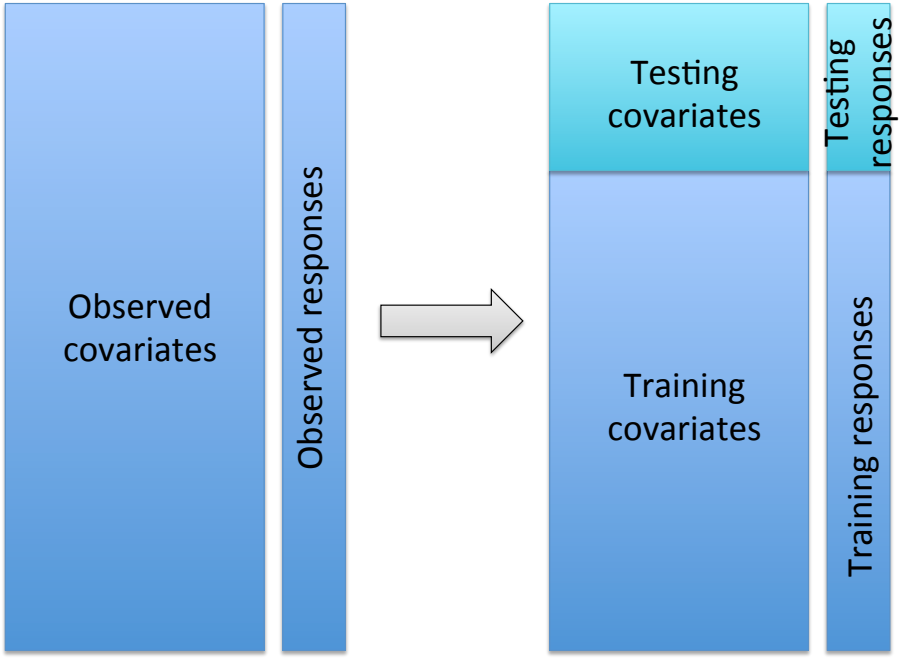
Relevant questions:

- What function will minimize this?
- Is this function a good predictor?

# Mean Squared Error

How do we estimate generalization error?

1) Break samples into training set and testing set (and possibly validation)

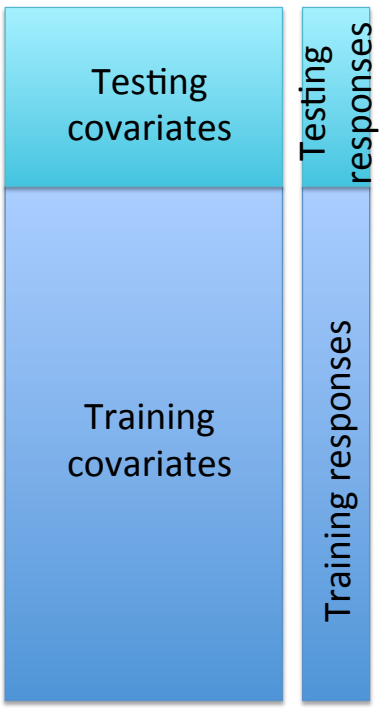



(it is usually a good idea to randomize the order of the data first)

# Mean Squared Error

How do we estimate generalization error?

2) Fit function on training set
3) Apply it to test set and estimate error



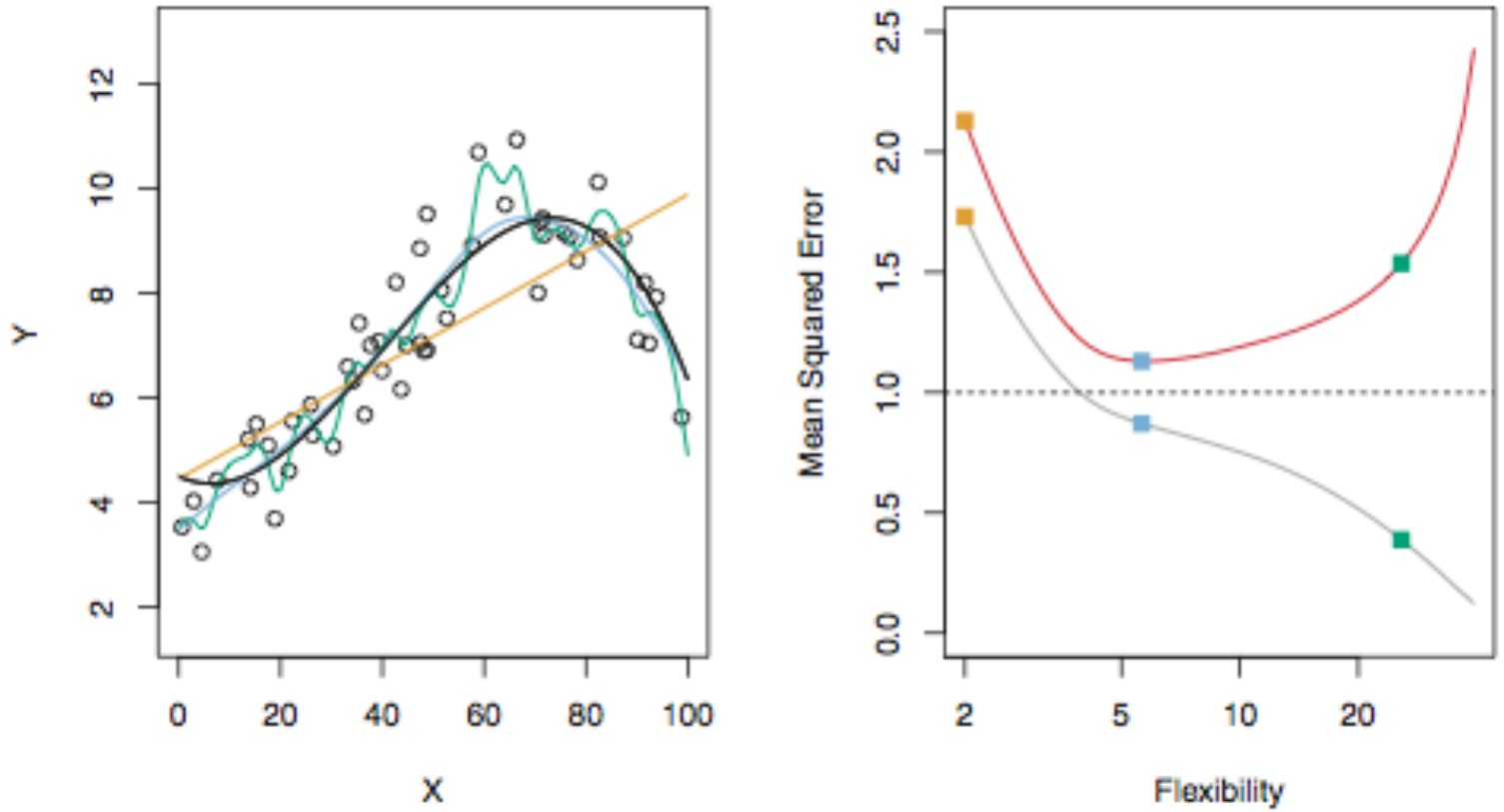

Test fitted function on testing data

$$MSE(\text{testing set}) = \frac{1}{n_{test}} \sum_{i=1}^{n_{test}} \left( y_i - \hat{f}_k(x_i) \right)^2$$

$$\hat{f}_k(x) = \frac{1}{k} \sum_{i \text{ is in } \mathrm{N}_k(x)} y_i$$

Fit function on training data

# Mean Squared Error




Why do you think the testing MSE has a U shape? What about the training MSE?[1]

[1] Some images taken from *An Introduction to Statistical Learning* by James, Witten, Hastie and Tibshirani.

# Outline

# Bias-Variance Decomposition

Training data $\mathcal{T} = \{(x_i, y_i)\}_{i=1}^{n}$.

- What is the mean squared error between the true $f(x_0)$ and the approximated value for a fixed $x_0$?
- Important: $\hat{y} = \hat{f}(x_0)$ is a random variable here (determined by noisy $\mathcal{T}$)
- Hence the expected reducible error, $\mathbb{E}$ over $\mathcal{T}$, is:

$$\begin{aligned} MSE(x_0) &= E[f(x_0) - \hat{f}(x_0)]^2 \\ &= E\left[\hat{f}(x_0) - E[\hat{f}(x_0)] + E[\hat{f}(x_0)] - f(x_0)\right]^2 \\ &\dots \end{aligned}$$

# Bias-Variance Decomposition

$$
\begin{aligned}
MSE(x_0) &= E[f(x_0) - \hat{f}(x_0)]^2 \\
&= E\left[\hat{f}(x_0) - E[\hat{f}(x_0)] + E[\hat{f}(x_0)] - f(x_0)\right]^2 \\
&= E\left[\hat{f}(x_0) - E[\hat{f}(x_0)]\right]^2 + E\left[E[\hat{f}(x_0)] - f(x_0)\right]^2 \\
&\qquad +2E\left[\hat{f}(x_0) - E[\hat{f}(x_0)]\right] E\left[E[\hat{f}(x_0)] - f(x_0)\right] \\
&= E\left[\hat{f}(x_0) - E[\hat{f}(x_0)]\right]^2 + E\left[E[\hat{f}(x_0)] - f(x_0)\right]^2 \\
&= E\left[\hat{f}(x_0) - E[\hat{f}(x_0)]\right]^2 + \left(E[\hat{f}(x_0)] - f(x_0)\right)^2 \\
&= \mathrm{Var}(\hat{f}(x_0)) + \mathrm{Bias}^2(\hat{f}(x_0))
\end{aligned}
$$

*Generalization error is estimator variance plus bias squared!*
*Note that all these terms are nonnegative!*

# Bias-Variance Decomposition

Bias: $E[\hat{f}(x_0)] - f(x_0)$

- error caused by difference between expected estimator and true function
- we say an estimator is *unbiased* if $E[\hat{f}(x_0)] - f(x_0) = 0$
- (estimate a mean with $\frac{1}{n}\sum z_i$ and $\frac{1}{n+100}\sum z_i$ )

Variance: $E\left[\hat{f}(x_0) - \mathbb{E}[\hat{f}(x_0)]\right]^2$

- this is a measure of *estimator spread*... that is, how much does the estimator vary with a new data set?

Reducible error is:

$$E[f(x_0) - \hat{f}(x_0)]^2 = \mathrm{Var}(\hat{f}(x_0)) + \mathrm{Bias}^2(\hat{f}(x_0))$$

# Bias-Variance Decomposition

Compare the predicted value with the actual value...
(remember $y = f(x) + \epsilon$)

$$
\begin{aligned}
E[y_0 - \hat{f}(x_0)]^2 &= E[f(x_0) + \epsilon - \hat{f}(x_0)]^2 \\
&= E[f(x_0) - \hat{f}(x_0)]^2 + E[\epsilon]^2 + 2E\left[\epsilon(f(x_0) - \hat{f}(x_0))\right] \\
&= \mathrm{Var}(\hat{f}(x_0)) + \mathrm{Bias}^2(\hat{f}(x_0)) + \mathrm{Var}(\epsilon)
\end{aligned}
$$

MSE = estimator variance + estimator bias$^2$ + noise variance
MSE = reducible + irreducible error

# Bias-Variance Decomposition

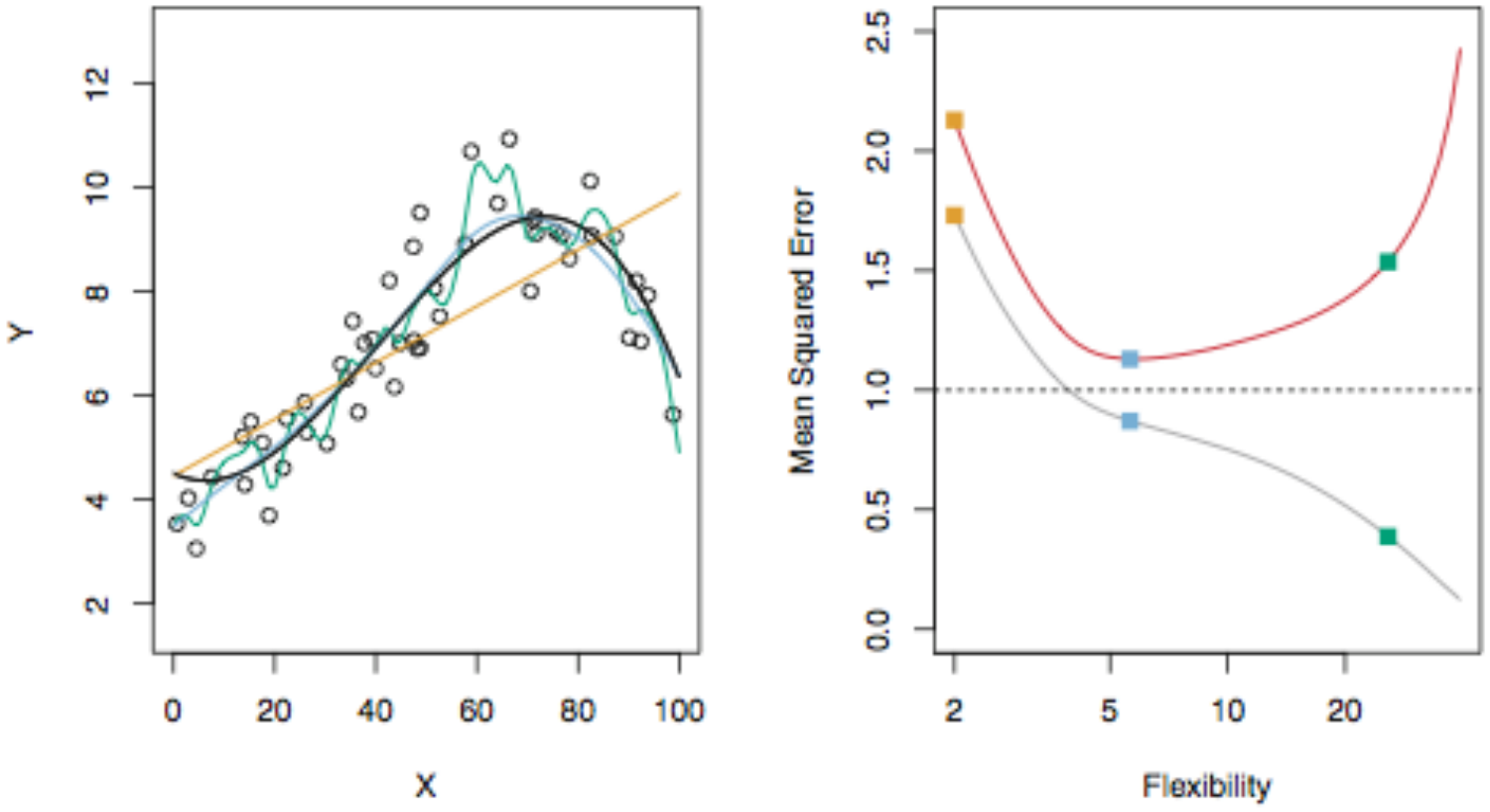
Let's think about estimators for a minute and answer some questions:

- Do **more flexible** estimators have more or less *bias* than **less flexible** estimators?
- Do **more flexible** estimators have more or less *variance* than **less flexible** estimators?
- Does **more data** increase or reduce the *bias* of an estimator?
- Does **more data** increase or reduce the *variance* of an estimator?

What rules of thumb can we get about data size? about model complexity?

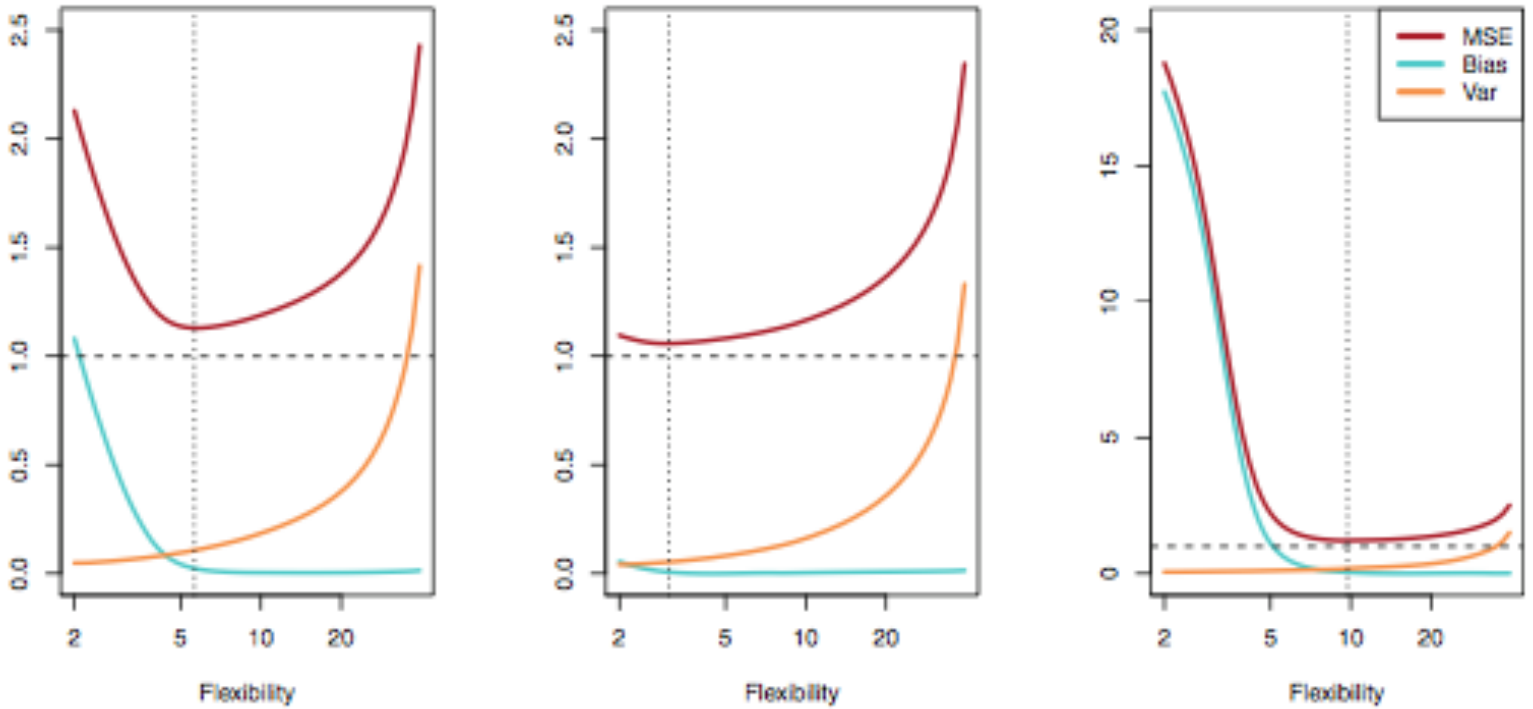
# Mean Squared Error

Let's look at this picture again. Is there any explanation in terms of estimator bias and variance?



# Mean Squared Error

Here is a breakdown of MSE in terms of estimator bias, estimator variance and noise variance.



# Bias-Variance Decomposition: One more time

Compare the predicted value with the actual value...
(remember $y = f(x) + \epsilon$)

$$
\begin{aligned}
& E[y_0 - \hat{f}(x_0)]^2 \\
= \; & E[f(x_0) + \epsilon - \hat{f}(x_0)]^2 \\
= \; & E[f(x_0) - \hat{f}(x_0)]^2 + E[\epsilon]^2 \\
= \; & E\left[\hat{f}(x_0) - E[\hat{f}(x_0)]\right]^2 + \left(E[\hat{f}(x_0)] - f(x_0)\right)^2 + E(\epsilon^2) \\
= \; & \mathrm{Var}(\hat{f}(x_0)) + \mathrm{Bias}^2(\hat{f}(x_0)) + \mathrm{Var}(\epsilon)
\end{aligned}
$$

MSE = estimator variance + estimator bias$^2$ + noise variance
MSE = reducible + irreducible error

# Outline

# Classification Generalization Error

MSE is not a great metric for classification. (Errors not Gaussian!)
Instead, we estimate the *error rate*, or the proportion misclassified:

$$Err = \mathbb{E}\left[\mathbf{1}_{\{Y_0 \neq \hat{Y}_0\}}\right]$$

Here $\mathbf{1}_A$ (denoted $I(A)$ in book) is an *indicator function*:

$$\mathbf{1}_A = \begin{cases} 1 & \text{if } A \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

Training error rate:

$$Err_{training} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{y_i \neq \hat{y}_i}$$

# The Bayes Classifier

It can be shown that the classifier that minimizes

$$Err = \mathbb{E}\left[\mathbf{1}_{\{Y_0 \neq \hat{Y}_0\}}\right]$$

assigns the predictive label to the class with the highest conditional probability:

$$\hat{Y}(x_0) = \arg\max \mathbb{P}\left(Y = j \,|\, X = x_0\right).$$

This is called the *Bayes Classifier*. The error rate is

$$1 - \mathbb{E}\left(\max_j \mathbb{P}(Y = j \,|\, X)\right).$$

Why can't we just use the Bayes classifier?

# The Bayes Classifier

Trying to get $\mathbb{P}(Y = j \,|\, X = x_0)$:

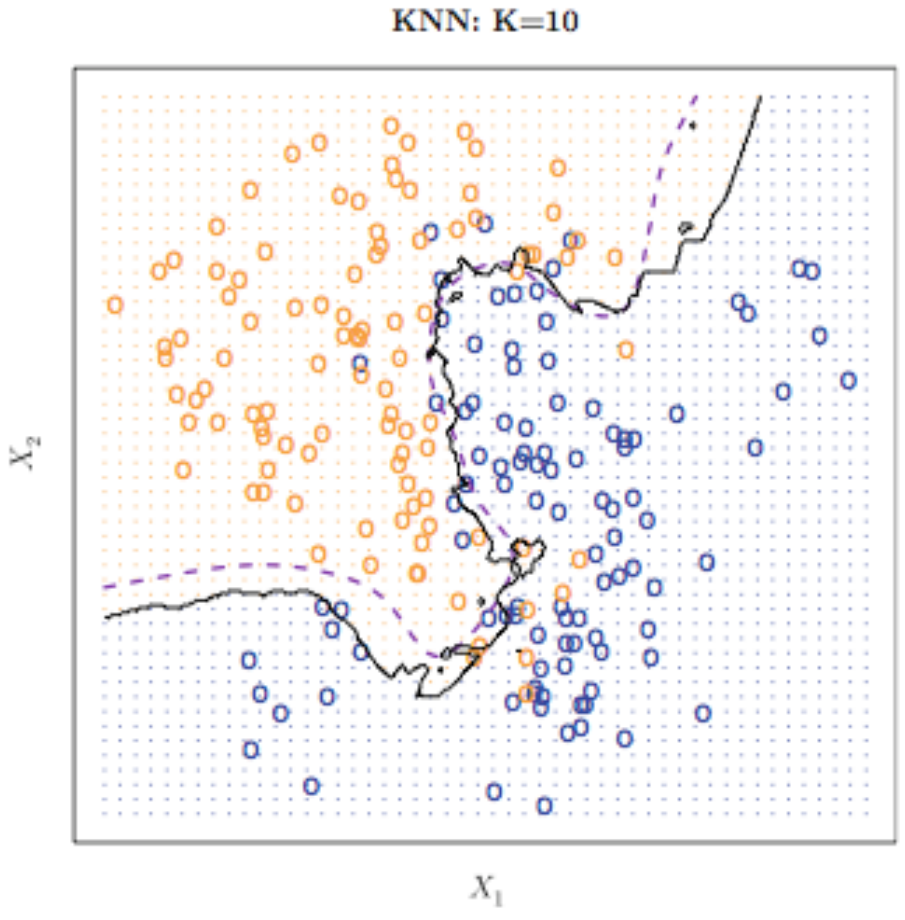
- can't really condition on event $\{X = x_0\}$
- ...but we can condition on event $\{X \text{ near } x_0\}$
- ...so let's choose the $k$ closest neighbors to $x_0$

$$\mathbb{P}(Y = j \,|\, X = x_0) \approx \frac{1}{k} \sum_{i \in N_k(x_0)} \mathbf{1}_{\{y_i = j\}}$$

For classification, kNN is an approximation to the Bayes classifier!

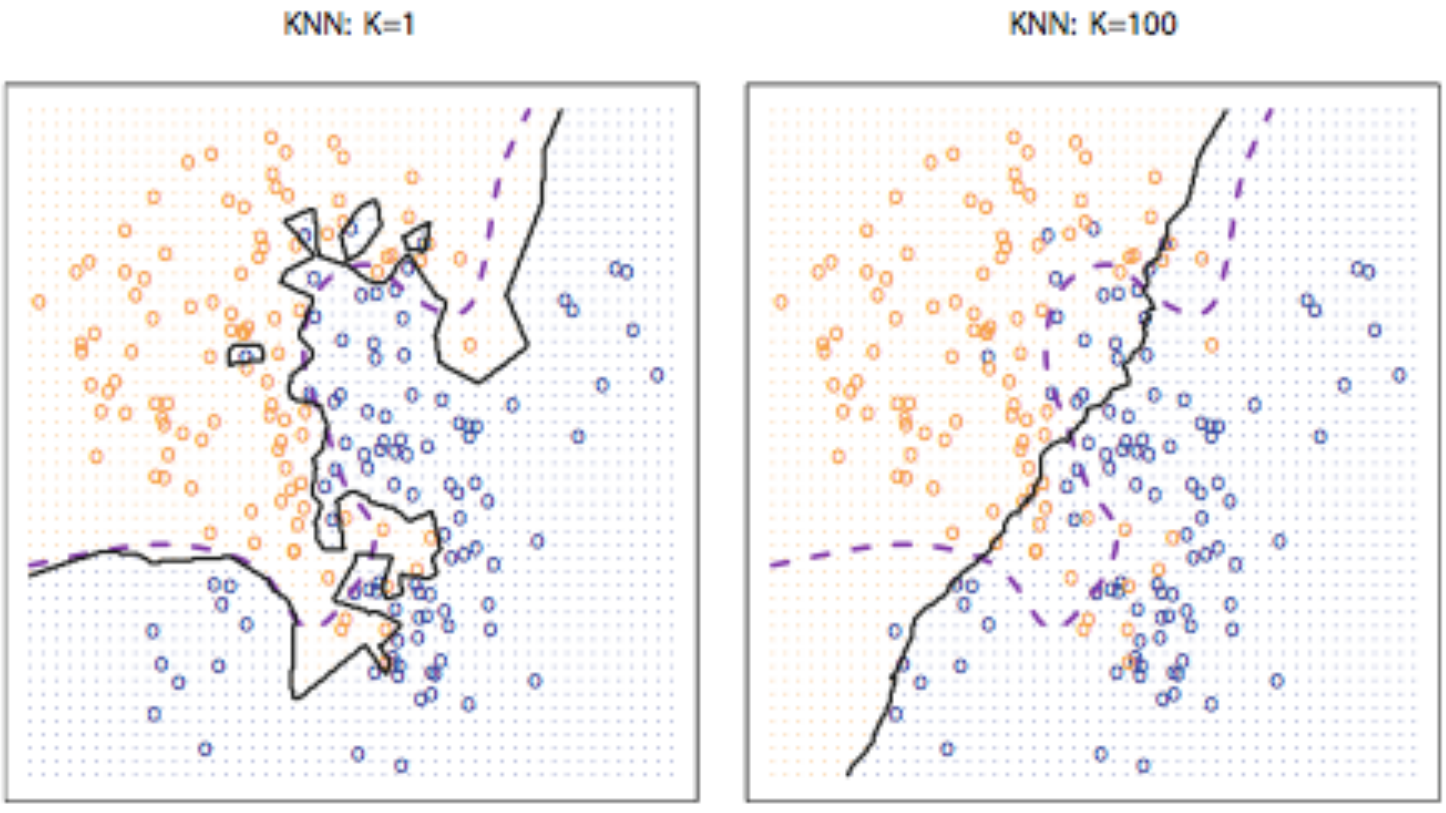
# The Bayes Classifier and kNN

Let's compare the classification boundary produced by kNN and the Bayes classifier:



# The Bayes Classifier and kNN

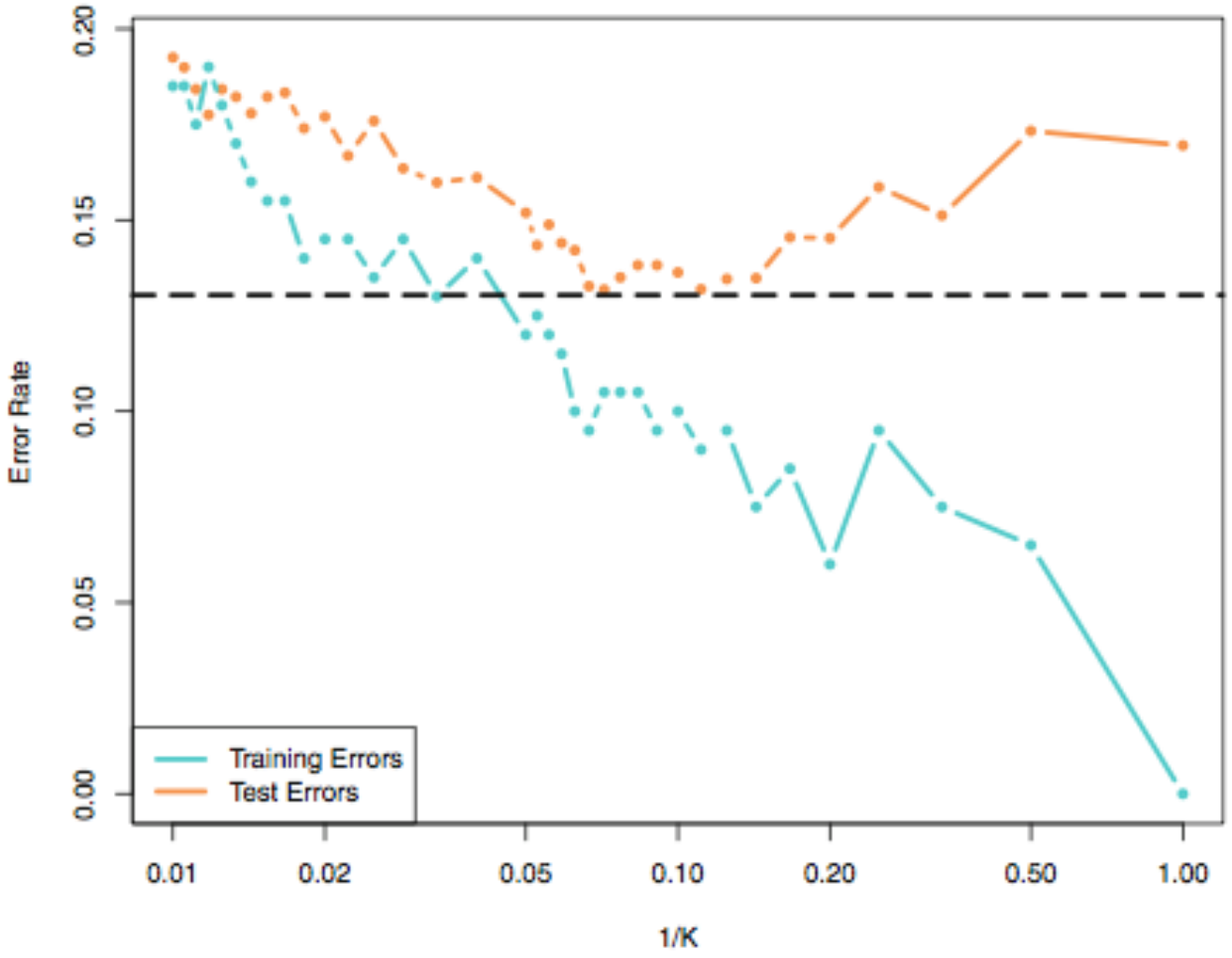
What happens if we change $k$?




What happens to the bias? Variance?

# The Bayes Classifier and kNN

What happens if we change $k$?




Can you explain this in terms of bias and variance?

# Outline

# Fun with kNN: Image Processing

- This is a $512 \times 512$ pixel portable networks graphics (.png) image
- Each pixel has an associated RGB (red, green, blue) value taking 1 of 256 values
- Stored as a $512 \times 512 \times 3$ array

# Image Processing

Umm... how do I get that into R?

- various graphics packages, some focused on different image types
- I used the package png

```
> library(png)
> lena.im <- readPNG("Lenna.png")
> dim(lena.im)
[1] 512 512   3
```

# Image Processing

```
> lena.im[1:5,1:5,]
, , 1

          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.8862745 0.8862745 0.8745098 0.8745098 0.8862745
[2,] 0.8862745 0.8862745 0.8745098 0.8745098 0.8862745
[3,] 0.8862745 0.8862745 0.8745098 0.8745098 0.8862745
[4,] 0.8862745 0.8862745 0.8745098 0.8745098 0.8862745
[5,] 0.8862745 0.8862745 0.8745098 0.8745098 0.8862745

, , 2

          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.5372549 0.5372549 0.5372549 0.5333333 0.5411765
[2,] 0.5372549 0.5372549 0.5372549 0.5333333 0.5411765
[3,] 0.5372549 0.5372549 0.5372549 0.5333333 0.5411765
[4,] 0.5372549 0.5372549 0.5372549 0.5333333 0.5411765
[5,] 0.5372549 0.5372549 0.5372549 0.5333333 0.5411765

, , 3

          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.4901961 0.4901961 0.5215686 0.5019608 0.4705882
[2,] 0.4901961 0.4901961 0.5215686 0.5019608 0.4705882
[3,] 0.4901961 0.4901961 0.5215686 0.5019608 0.4705882
[4,] 0.4901961 0.4901961 0.5215686 0.5019608 0.4705882
[5,] 0.4901961 0.4901961 0.5215686 0.5019608 0.4705882
```

# Image Processing

Image reconstruction:

- Pixels are randomly removed (here 25%)
- How can we reconstruct the original image?

# Image Processing

Lena: original, 1nn

# Image Processing

Lena: original, 1nn, 3nn, 5nn (clockwise from upper left)

# Image Processing

Lena: original, 1nn, 10nn, 100nn (clockwise from upper left)

# Image Processing

Thought experiment:

- what would happen if we used *all* of the pixels as a neighborhood?
- would this reconstruction look better or worse to you? why?
- and why does kNN work so well in this case?

# Image Processing

Homework 3:

- you will be using $k$NN for face recognition
- plan:
  1. use PCA to transform high dimensional image space to something smaller
  2. represent image by a small number of scores
  3. use 1NN in score space to classify new image (who is closest?)
- want to know:
  1. what is the expected predictive error for this classifier?
  2. when does it work well? poorly?
  3. why are all of the photos cropped?

*One last note about Lena...*