S&DS 355 / 365 / 565

**Data Mining and Machine Learning**

# Bias, Variance and Cross Validation

Tuesday, September 17th

Yale

# Outline

- Finish off SGD
- Bias/variance redux
- Cross validation

# SGD for general loss

SGD update:

$$\boldsymbol{\beta} \longleftarrow \boldsymbol{\beta} - \eta \nabla L(y, \boldsymbol{\beta}^T x)$$

$$\beta_j \longleftarrow \beta_j - \eta \frac{\partial L(y, \boldsymbol{\beta}^T x)}{\partial \beta_j}$$

- $\eta$ is the *learning rate* or "step size"
- Needs to be chosen carefully, getting smaller over time

# SGD: choice of learning rate

A conservative choice of learning rate is

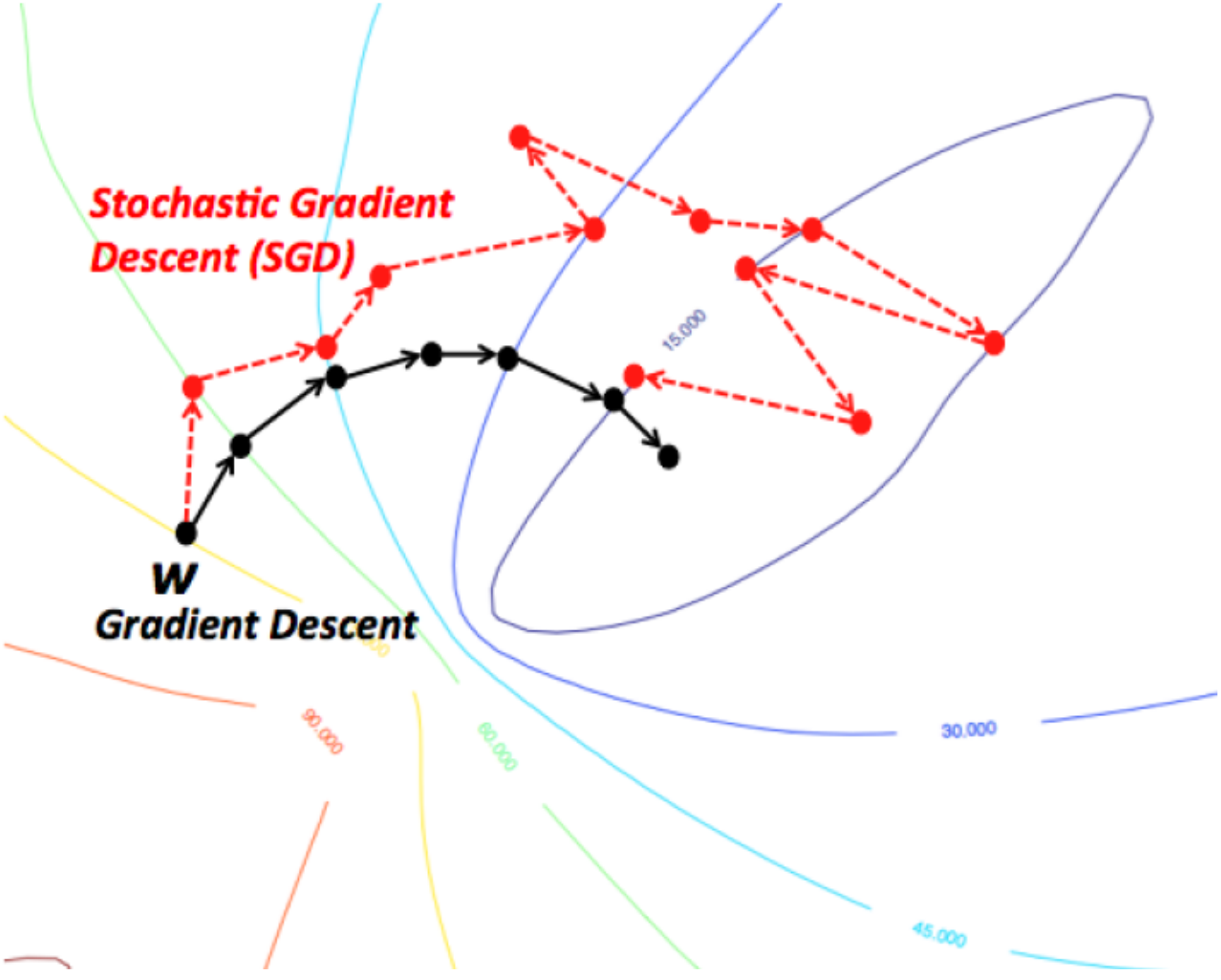
$$\eta_t = \frac{1}{t}$$

A more agressive choice is

$$\eta_t = \frac{1}{\sqrt{t}}$$

*Which is more appropriate for GD?*
*Which is more appropriate for SGD?*

# Diagramatic Differences



# SGD: choice of learning rate

Learning rate should scale as

$$\eta_t = \frac{1}{\sqrt{t}}$$

Problem: Some of the updates may be on different scales.

# SGD: choice of learning rate

Learning rate should scale as

$$\eta_t = \frac{1}{\sqrt{t}}$$

Problem: Some of the updates may be on different scales.

Solution: Let $g_{tj} = \frac{\partial L(y_t, \beta^T x_t)}{\partial \beta_j}$

Scale gradients to get update rule

$$\beta_j \longleftarrow \beta_j - \eta \frac{g_{tj}}{\sqrt{\sum_{s=1}^{t} g_{sj}^2}}$$

# SGD: scaling issues

For a linear model, the SGD update is

$$\beta_j \longleftarrow \beta_j - c_t x_j$$

If $x_j$ increases by a factor of two, the weight $\beta_j$ should decrease by a factor of two.

This update doesn't respect that scaling

# SGD: scaling issues

Usual solution is to "standardize" each variable — subtract out the mean and divide by the standard deviation

$$x_j \leftarrow \frac{x_j - \text{mean}(x_j)}{\sqrt{\text{var}(x_j)}}$$

But this involves "looking ahead" to compute the mean and variance, and destroys the online property of the algorithm

# SGD: scaling issues

Usual solution is to "standardize" each variable — subtract out the mean and divide by the standard deviation

$$x_j \leftarrow \frac{x_j - \text{mean}(x_j)}{\sqrt{\text{var}(x_j)}}$$

But this involves "looking ahead" to compute the mean and variance, and destroys the online property of the algorithm

Solution: The mean and variance can be updated in an online manner, in constant time, by storing auxiliary variables for each component $j$.

# SGD: Regularization

A "ridge" penalty $\lambda \sum_{j=1}^{d} \beta_j^2$ is easily handled.

Gradient changes by an additive term $2\lambda\beta_j$. Update becomes

$$\begin{aligned} \beta_j &\longleftarrow \beta_j + \eta\{(y-\pi)x_j - \lambda\beta_j\} \\ &= (1-\eta\lambda)\beta_j + \eta(y-\pi)x_j \end{aligned}$$

$$\beta_j x_j \longleftarrow (1-\eta\lambda)\beta_j x_j + \eta(y-\pi)x_j^2$$

Observe that this "does the right thing" whether $\beta_j$ wants to be large positive or negative.

- *The penalty shrinks $\beta_j$ toward zero*

# Batch GD vs. SGD

- In a “batch” algorithm, we compute the exact gradient by summing over the entire training set
- In a “stochastic” algorithm, we tolerate noise in the estimate of the gradient, in exchange for speed
- Could also compute the gradient over “mini batches”
- Note: SGD is (apparently) difficult to parallelize

# SGD: Linear regression

Batch gradient descent update step (using all $\{(x_i, y_i)\}_{i=1}^n$ observations):

$$\beta_j \leftarrow \beta_j + \rho \sum_{i=1}^{n} (y_i - x_i^t \beta) x_{ij}.$$

Stochastic gradient descent update (using a single observation $(x, y)$):

$$\beta_j \leftarrow \beta_j + \rho (y - x^t \beta) x_j.$$

# SGD: Linear regression

Batch gradient descent update step (using all $\{(x_i, y_i)\}_{i=1}^n$ observations):

$$\beta_j \leftarrow \beta_j + \rho \sum_{i=1}^{n} (y_i - x_i^t \beta) x_{ij}.$$

Stochastic gradient descent update (using a single observation $(x, y)$):

$$\beta_j \leftarrow \beta_j + \rho (y - x^t \beta) x_j.$$

Equivalently,

$$\beta_j \leftarrow \beta_j + \rho (y - \widehat{y}) x_j.$$

# SGD: Linear regression

$$\beta_j \leftarrow \beta_j + \rho(y - \widehat{y})x_j.$$

In words:

- Initialize all coefficients $\beta_j$ to 0.
- For observation $x$:
  - Predict $\widehat{y} = x^t\beta$.
  - Observe true response $y$.
  - Update $\beta$ so that $\widehat{y}$ is closer to $y$.

# Stochastic Gradient Descent

Key points:

- Randomize data beforehand
- $\rho$ (the learning rate) needs to be selected in a way that decreases with time
  - e.g. $\rho_t = \frac{1}{t}$, $\rho_t = \frac{1}{\sqrt{t}}$
- Scaling data can ensure that variance of variables don't interfere

$$x_j \leftarrow \frac{x_j - \text{mean}(x_j)}{\sqrt{\text{var}(x_j)}}$$

- Takes a windier path, and may not actually converge to the global minimum.

# Model Selection

For purposes of prediction, **minimizing test error** is priority.

Recall our two error metrics for evaluating predictions $\widehat{f}(x_i)$:

- Regression:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{f}(x_i))^2$$

- Classification:

$$Err = \frac{1}{n}\sum_{i=1}^{n}\mathbb{1}\left\{\widehat{f}(x_i) \neq y_i\right\}$$

# Bias-Variance Tradeoff

(Regression case)

Given $Y = f(X) + \epsilon$, where $E(\epsilon) = 0$ and $Var(\epsilon) = \sigma^2$, consider a predictor $\widehat{f}$.

Expected MSE for predicting a new $Y$ at $X = x$ can be decomposed into:

$$E[(Y - \widehat{f}(x))^2] = Var(\widehat{f}(x)) + [Bias(\widehat{f}(x))]^2 + \sigma^2$$

# Bias-Variance Tradeoff

$$E[(Y - \widehat{f}(x))^2] = Var(\widehat{f}(x)) + [Bias(\widehat{f}(x))]^2 + \sigma^2$$

- $Var(\widehat{f})$ is the amount of variability in our predictor with different training set.
- $Bias(\widehat{f})$ is the systematic error introduced by model approximation.
- $\sigma^2$ is *irreducible error*, inherent in the error term $\epsilon$.

# Bias-Variance Tradeoff

$$E[(Y - \widehat{f}(x))^2] = Var(\widehat{f}(x)) + [Bias(\widehat{f}(x))]^2 + \sigma^2$$

- $Var(\widehat{f})$ is the amount of variability in our predictor with different training set. Increases with increasing model flexibility.
- $Bias(\widehat{f})$ is the systematic error introduced by model approximation. Decreases with increasing model flexibility.
- $\sigma^2$ is *irreducible error*, inherent in the error term $\epsilon$. Cannot get rid of this!

Need to balance bias and variance.

# Bias-Variance

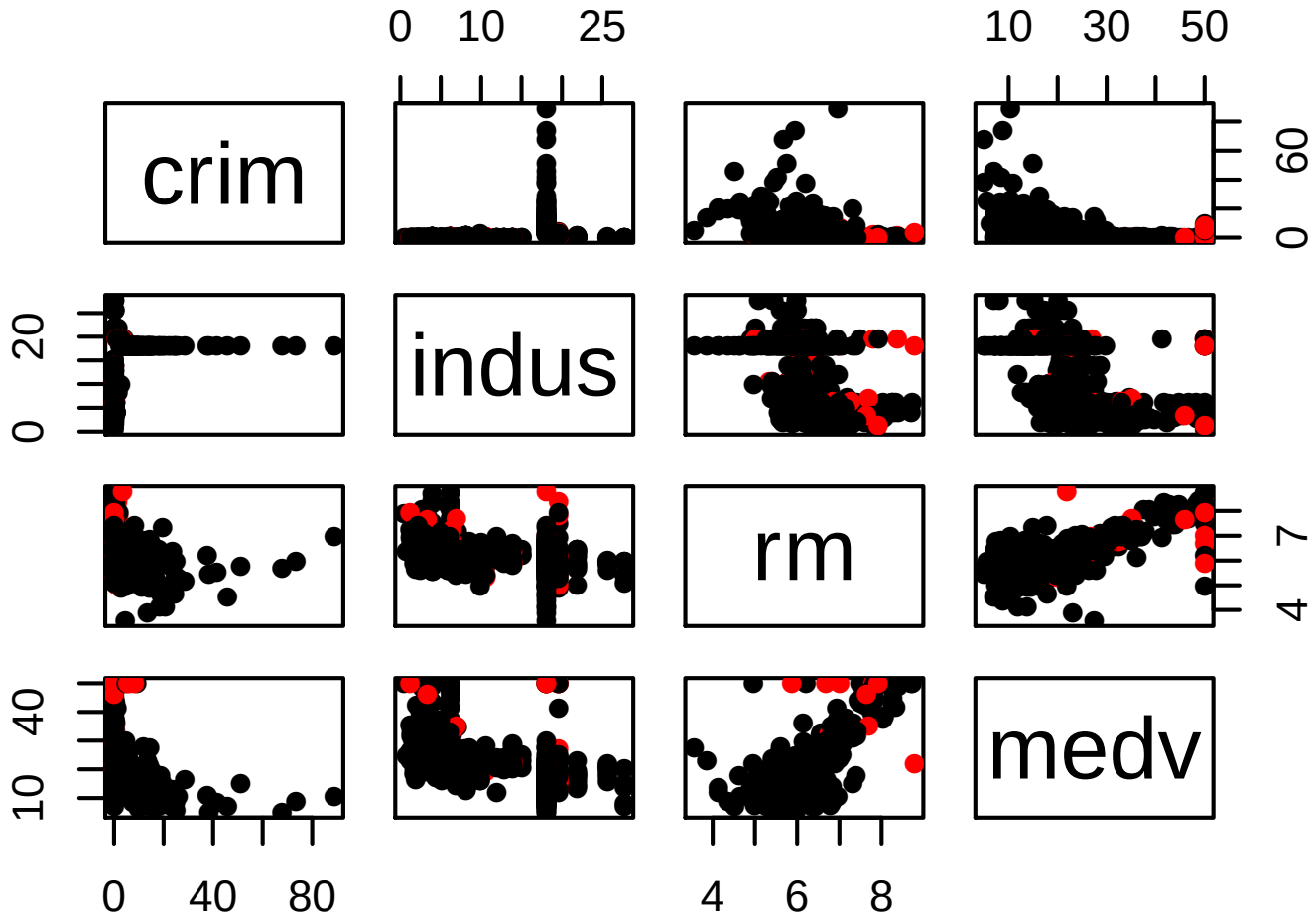
How does this apply to the $k$-NN algorithm?

# Cross-Validation

**Cross-validation** is an intuitive, widely-applicable approach for:

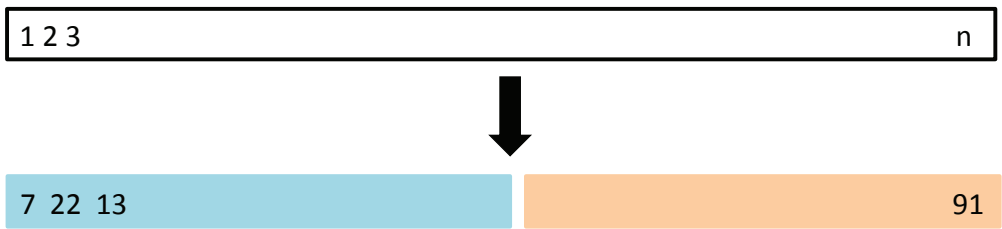
- model assessment
- model selection

# Example: Boston Housing



# Validation Sets

We've been doing this:




1. Divide dataset randomly into a training set and a validation set.
2. Fit the model on the training set.
3. Use the validation set to obtain estimated test error.
4. Repeat!

# Validation Sets

```
set.seed(365)
train <- sample(1:nrow(Boston), nrow(Boston)/2, replace=FALSE)
test <- Boston[-train,]
train <- Boston[train,]
m1 <- lm(medv ~ lstat, data=train)
p1 <- predict(m1, test)
```

# Validation Sets

Example:

$$\widehat{medv} = \widehat{\beta}_0 + \widehat{\beta}_1 lstat$$

Estimated test error:

```
mean((p1 - test$medv)^2)
```

```
## [1] 43.57431
```

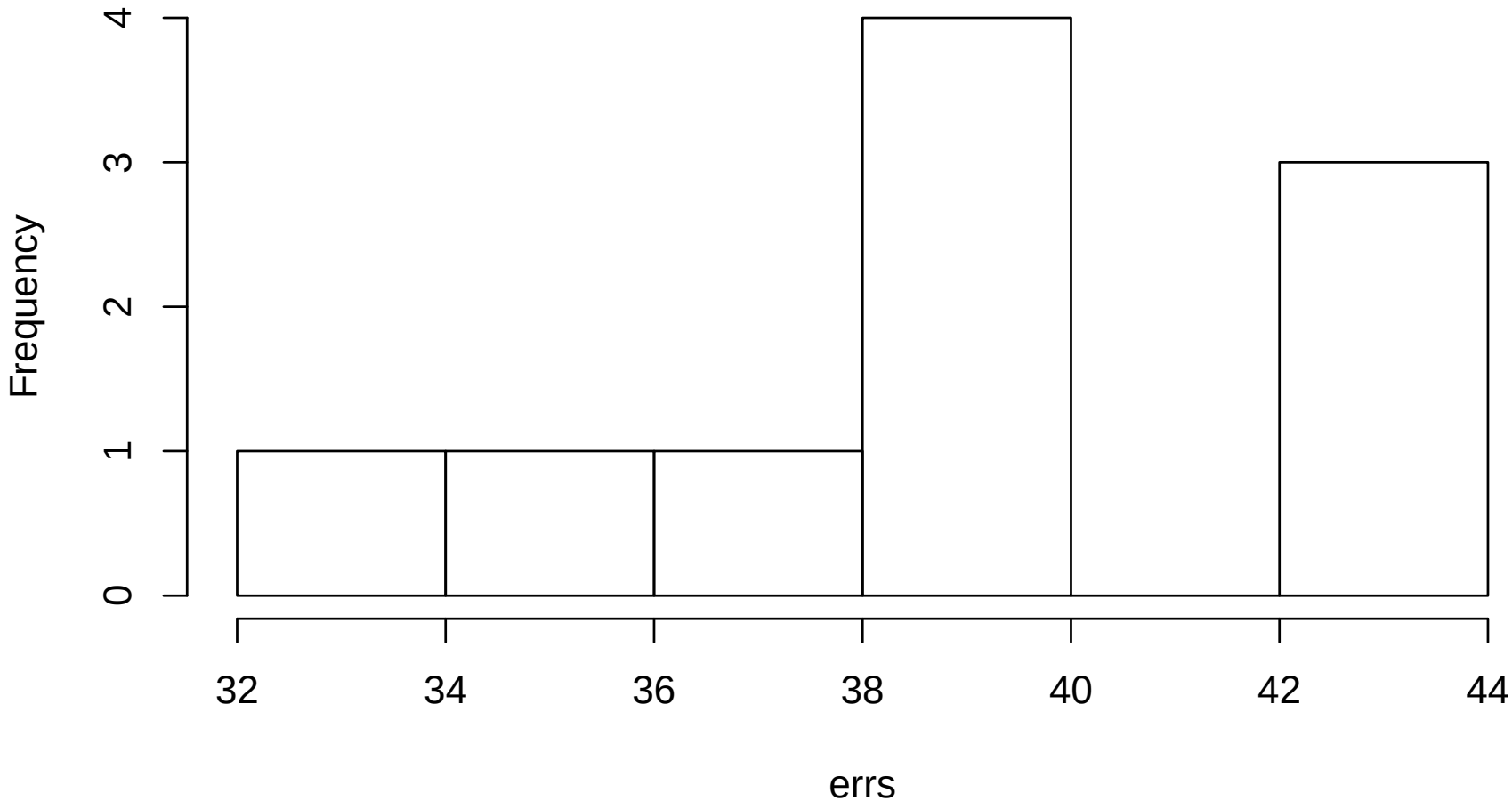
# Validation Sets

Repeat x 10:

```
set.seed(665)
errs <- rep(NA, 10)
for (i in 1:10) {
  train <- sample(1:nrow(Boston), nrow(Boston)/2, replace=FALSE)
  test <- Boston[-train,]
  train <- Boston[train,]
  m1 <- lm(medv ~ lstat, data=train)
  summary(m1)
  p1 <- predict(m1, test)
  errs[i] <- mean((p1 - test$medv)^2)
}
```

# Histogram of errors

```
hist(errs, main="")
```



# Validation Sets

- highly variable validation error
- only uses a fraction of the training set

# Leave-One-Out Cross-Validation

How do we use more data to train with?

- Use a tiny validation set (e.g. $(x_1, y_1)$)
- Train with the rest (e.g. $\{(x_2, y_2), \ldots, (x_n, y_n)\}$)

How do we feel about error rate evaluated on a single observation?

# Leave-One-Out Cross-Validation

How do we use more data to train with?

- Use a tiny validation set (e.g. $(x_1, y_1)$)
- Train with the rest (e.g. $\{(x_2, y_2), \ldots, (x_n, y_n)\}$)

How do we feel about error rate evaluated on a single observation?

Not good, but we can iterate through the dataset, each time using a different $(x_i, y_i)$ as the validation set and obtaining an error $MSE_i$.

# Leave-One-Out Cross-Validation

| | Iteration | | | | | |
|---|---|---|---|---|---|---|
| Obs | 1 | 2 | 3 | 4 | ... | n |
| 1 | valid | train | train | train | ... | train |
| 2 | train | valid | train | train | ... | train |
| 3 | train | train | valid | train | ... | train |
| 4 | train | train | train | valid | ... | train |
| ... | ... | ... | ... | ... | ... | ... |
| $n$ | train | train | ... | ... | ... | valid |
| MSE | $MSE_1$ | $MSE_2$ | $MSE_3$ | $MSE_4$ | ... | $MSE_n$ |

# Leave-One-Out Cross-Validation

LOOCV estimate of test error is given by:

$$CV_{(n)} = \frac{1}{n} \sum_i MSE_i$$

# Leave-One-Out Cross-Validation

LOOCV estimate of test error is given by:

$$CV_{(n)} = \frac{1}{n} \sum_i MSE_i$$

For our example, a single number:

```
##        1
## 38.8901
```

# k-fold Cross-Validation

A potentially faster approach:

- Randomly divide the dataset into $k$ *folds*.
- For $b = 1, \ldots, k$:
  - Use $b$-th fold ("batch") as validation set.
  - Use everything else as training set.
  - Compute validation error on $b$-th fold.
- Estimate test error using:

$$CV_{(k)} = \sum_b \frac{n_b}{n} MSE_b,$$

where $n_b$ is the total # observations in the $b$-th fold, and $n$ is the total # observations in the entire dataset.

# k-fold Cross-Validation

| | Iteration | | | | | | |
|---|---|---|---|---|---|---|---|
| Obs | 1 | 2 | 3 | 4 | ... | $k$ | |
| 1 | valid | train | train | train | ... | train | fold 1 |
| 2 | valid | train | train | train | ... | train | |
| 3 | valid | train | train | train | ... | train | |
| 4 | train | valid | train | train | ... | train | |
| ... | ... | ... | ... | ... | ... | ... | |
| $n-2$ | train | train | ... | ... | ... | valid | fold $v$ |
| $n-1$ | train | train | ... | ... | ... | valid | |
| $n$ | train | train | ... | ... | ... | valid | |
| MSE | $MSE_1$ | $MSE_2$ | $MSE_3$ | $MSE_4$ | ... | $MSE_k$ | |

# k-fold Cross-Validation

| | Iteration | | | | | | |
|---|---|---|---|---|---|---|---|
| Obs | 1 | 2 | 3 | 4 | ... | $k$ | |
| 1 | valid | train | train | train | ... | train | fold 1 |
| 2 | valid | train | train | train | ... | train | |
| 3 | valid | train | train | train | ... | train | |
| 4 | train | valid | train | train | ... | train | |
| ... | ... | ... | ... | ... | ... | ... | |
| $n-2$ | train | train | ... | ... | ... | valid | fold $v$ |
| $n-1$ | train | train | ... | ... | ... | valid | |
| $n$ | train | train | ... | ... | ... | valid | |
| MSE | $MSE_1$ | $MSE_2$ | $MSE_3$ | $MSE_4$ | ... | $MSE_k$ | |

$n$-fold CV is just LOOCV.

# k-fold Cross-Validation

Running $k = 5$ and $k = 10$:

```
set.seed(665)
kvals <- c(5, 10)
errs <- rep(0, length(kvals))
z <- Boston[sample(1:nrow(Boston)),]
for (j in 1:length(kvals)) {
  k <- kvals[j]
  folds <- as.numeric(cut(1:nrow(z), k))
  for (i in 1:k) {
    train <- z[folds != i,]
    valid <- z[folds == i,]
    m1 <- lm(medv ~ lstat, data=train)
    p1 <- predict(m1, test)
    errs[j] <- errs[j] +
         sum(folds==i)/nrow(z)*mean((p1-test$medv)^2)
  }
}
errs[1]/nrow(z)
```

```
## [1] 0.4546264
```

# k-fold Cross-Validation

Estimated error for $k = 5$ and $k = 10$:

```
errs[1]/nrow(z)
```

```
## [1] 0.4546264
```

```
errs[2]/nrow(z)
```

```
## [1] 0.4546724
```

# A slick shortcut

Suppose that the fitted values can be written $\widehat{Y} = LY$ where $L$ is an $n \times n$ matrix.

This is the case for least squares regression

# A slick shortcut

Suppose that the fitted values can be written $\widehat{Y} = LY$ where $L$ is an $n \times n$ matrix.

This is the case for least squares regression

Then the leave-one-out-cross-validation error is

$$R_{LOO} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{Y}_{(-i)})^2$$

$$\frac{1}{n}\sum_{i=1}^{n}\left(\frac{Y_i - \widehat{Y}_i}{1 - L_{ii}}\right)^2$$

where $L_{ii}$ is the $i$th diagonal entry.

# A slick shortcut

Suppose that the fitted values can be written $\widehat{Y} = LY$ where $L$ is an $n \times n$ matrix.

This is the case for least squares regression

Then the leave-one-out-cross-validation error is

$$R_{LOO} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{Y}_{(-i)})^2$$

$$\frac{1}{n}\sum_{i=1}^{n}\left(\frac{Y_i - \widehat{Y}_i}{1 - L_{ii}}\right)^2$$

where $L_{ii}$ is the $i$th diagonal entry.

*So, no need to fit n regressions!*

# Model Selection

So far, we've used it to estimate the test error. It's also useful for model selection.

# Model Selection

So far, we've used it to estimate the test error. It's also useful for model selection.

Suppose we are interested in comparing the following 3 models:

Model 1:

$$\widehat{medv} = \widehat{\beta}_0 + \widehat{\beta}_1 lstat$$

Model 2:

$$\widehat{log(medv)} = \widehat{\beta}_0 + \widehat{\beta}_1 lstat$$

Model 3:

$$\widehat{medv} = \widehat{\beta}_0 + \widehat{\beta}_1 lstat + \widehat{\beta}_2 lstat^2$$

# Model Selection

So far, we've used it to estimate the test error. It's also useful for model selection.

Suppose we are interested in comparing the following 3 models:

Model 1:

$$\widehat{medv} = \widehat{\beta}_0 + \widehat{\beta}_1 lstat$$

Model 2:

$$\widehat{log(medv)} = \widehat{\beta}_0 + \widehat{\beta}_1 lstat$$

Model 3:

$$\widehat{medv} = \widehat{\beta}_0 + \widehat{\beta}_1 lstat + \widehat{\beta}_2 lstat^2$$

We can use cross-validation to estimate the test error for each of these models, and select the model with the lowest test error.

# What did we learn today?

- Cross validation is a practical way of estimating the variability of test error. Used for model selection.
- Leave-one-out CV is the most important version of CV. Has a shortcut formula.