

Lab #3: Linear Regression and Model Selection

CS 109A, STAT 121A, AC 209A: Data Science

Fall 2016

Harvard

Announcements

- ...

Today's lab: Problem 1

- a) Multiple linear regression from scratch
 - Fit regression model
 - Score regression model
- b) Confidence intervals on model parameters
 - Analyze histograms of model parameters
 - Compute confidence intervals

Today's lab: Problem 1

- a) Multiple linear regression from scratch
 - Fit regression model
 - Score regression model
- b) Confidence intervals on model parameters
 - Analyze histograms of model parameters
 - Compute confidence intervals

Review:

numpy basics

Data and models as matrices

	Predictors		Response
	X_1	X_2	Y
obs. 1	3	4	3
obs. 2	2	1	0
obs. 3	5	2	8

$\longrightarrow X = \begin{pmatrix} 3 & 4 \\ 2 & 1 \\ 5 & 2 \end{pmatrix}, Y = \begin{pmatrix} 3 \\ 0 \\ 8 \end{pmatrix}$

- X is two dimensional array with shape (3, 2)
- Y is two dimensional array with shape (3, 1)

Data and models as matrices

- Model:

$$f(X_1, X_2) = \underbrace{w_1 X_1 + w_2 X_2 + c}_{\downarrow}$$
$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}, c$$

- w is a two-dimensional array of shape $(2, 1)$
- c is a float scalar

Data and models as matrices

- Predictions as matrix multiplication:

$$\hat{Y} = X \times w + c$$

.....


$$\hat{Y} = \begin{bmatrix} 3 & 4 \\ 2 & 1 \\ 5 & 2 \end{bmatrix} \times \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + \begin{bmatrix} c \\ c \\ c \end{bmatrix} = \begin{bmatrix} w_1 \times 3 + w_2 \times 4 + c \\ w_1 \times 2 + w_2 \times 1 + c \\ w_1 \times 5 + w_2 \times 2 + c \end{bmatrix}$$

Data and model as matrices

- Predictions as matrix multiplication:

$$\hat{Y} = X \times w + c$$

$$\hat{Y} = \begin{bmatrix} 3 & 4 & 1 \\ 2 & 1 & 1 \\ 5 & 2 & 1 \end{bmatrix} \times \begin{bmatrix} w_1 \\ w_2 \\ c \end{bmatrix} = \begin{bmatrix} w_1 \times 3 + w_2 \times 4 + c \\ w_1 \times 2 + w_2 \times 1 + c \\ w_1 \times 5 + w_2 \times 2 + c \end{bmatrix}$$



Combine coefs & intercepts into a single array
Append a column of ones

numpy: basic matrix operations

- Create column matrix of ones

```
# Create a column of 'n' ones  
one_col = np.ones((n, 1))
```

- Matrix multiplication

```
# Multiply matrix 'a' of size m x k  
#           and matrix 'b' of size k x s  
# Outputs a matrix of size m x s  
c = np.dot(a, b)
```

numpy: basic matrix operations

- Concatenating arrays

```
# Concatenate two arrays  
# 'a' of size m1 x k and  
# 'b' of size m2 x k, along rows  
# Outputs an array of size (m1+m2) x k  
c = np.concatenate((a, b), axis = 0)
```

```
# Concatenate two arrays  
# 'a' of size m x k1 and  
# 'b' of size m x k2, along columns  
# Outputs an array of size m x (k1+k2)  
c = np.concatenate((a, b), axis = 1)
```

Computing predictions in numpy

- Append column of ones and multiply

```
# Compute predictions using matrix multiplication  
# x: array of predictors of size n x d  
# w: array of coefficients and intercept of size (d+1) x 1  
  
# Append a column of one's to 'x'  
n = x.shape[0]  
ones_col = np.ones((n, 1))  
x = np.concatenate((x, ones_col), axis=1)  
  
# Multiply 'x' with 'w'  
y_pred = np.dot(x, w)
```

numpy: other useful operations

- Matrix transpose

```
# Transpose of a matrix 'a'  
a_tranpose = np.transpose(a)
```

- Matrix inverse

```
# Invert a square matrix 'a'  
a_inverse = np.linalg.inv(a)
```

Review:

Multiple Linear Regression

Least-squares Solution

- Training data:

$$X = \begin{bmatrix} X_{11} & \dots & X_{1d} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ X_{n1} & \dots & X_{nd} & 1 \end{bmatrix} \quad Y = \begin{bmatrix} Y_1 \\ \vdots \\ Y_n \end{bmatrix}$$

- Model parameters:

$$w = \begin{bmatrix} w_1 \\ \vdots \\ w_d \\ c \end{bmatrix}$$

Least-squares Solution

- Predictions:

$$\hat{Y}_i = \sum_{j=1}^d w_j X_{ij}$$

- Minimize Least-squares Loss:

$$\begin{aligned} L(w) &= \sum_{i=1}^n (\hat{Y}_i - Y_i)^2 \\ &= \sum_{i=1}^n \left(\sum_{j=1}^d w_j X_{ij} - Y_i \right)^2 \end{aligned}$$

Least-squares Solution

- Set derivatives to zero:

$$\frac{\partial L(w)}{\partial w_1} = 0 \quad \dots \quad \frac{\partial L(w)}{\partial w_d} = 0$$

- Solve system of linear equations

$$\begin{aligned} 2 \sum_{i=1}^n \left(\sum_{j=1}^d w_j X_{ij} - Y_i \right) X_{i1} &= 0 \\ &\vdots \\ 2 \sum_{i=1}^n \left(\sum_{j=1}^d w_j X_{ij} - Y_i \right) X_{id} &= 0 \end{aligned}$$

Least-squares Solution

- Formulating in matrix form:

$$X^{\top}(Xw - Y) = 0$$

or

$$X^{\top}Xw = X^{\top}Y$$

- Solution:

$$w = (X^{\top}X)^{-1}X^{\top}Y$$

Least-squares Solution

- Formulating in matrix form:

$$X^{\top}(Xw - Y) = 0$$

or

$$X^{\top}Xw = X^{\top}Y$$

- Solution:

What can go wrong?

$$w = (X^{\top}X)^{-1}X^{\top}Y$$

Evaluating regression model

- R^2 score:

$$R^2 = 1 - \frac{RSS}{TSS}$$

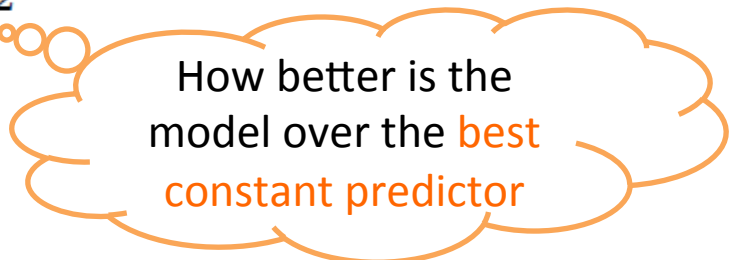
- Residual Sum of Squares (RSS)

$$RSS = \sum_{i=1}^n (\hat{Y}_i - Y_i)^2$$

- Total Sum of Squares (TSS)

$$TSS = \sum_{i=1}^n (\bar{Y} - Y_i)^2$$

where $\bar{Y} = \frac{1}{n} \sum_{i=1}^n Y_i$



How better is the model over the **best constant predictor**

Tasks

- Fit regression model to training set
 - `multiple_linear_regression_fit`
 - input: `x_train, y_train`
 - returns: `w, c`
- Evaluate model on test set
 - `multiple_linear_regression_score`
 - input: `w, c, x_test, y_test`
 - returns: `r_squared`

Today's lab: Problem 1

- a) Multiple linear regression from scratch
 - Fit regression model
 - Score regression model
- b) Confidence intervals on model parameters
 - Analyze histograms of model parameters
 - Compute confidence intervals

Subsampling

- Repeat 200 times: Random subsamples of size 100

X_1	X_2	Y
3	4	5
2	1	0
5	2	3
1	0	1
\vdots		
9	3	6
7	0	1

(x_subsample, y_subsample)



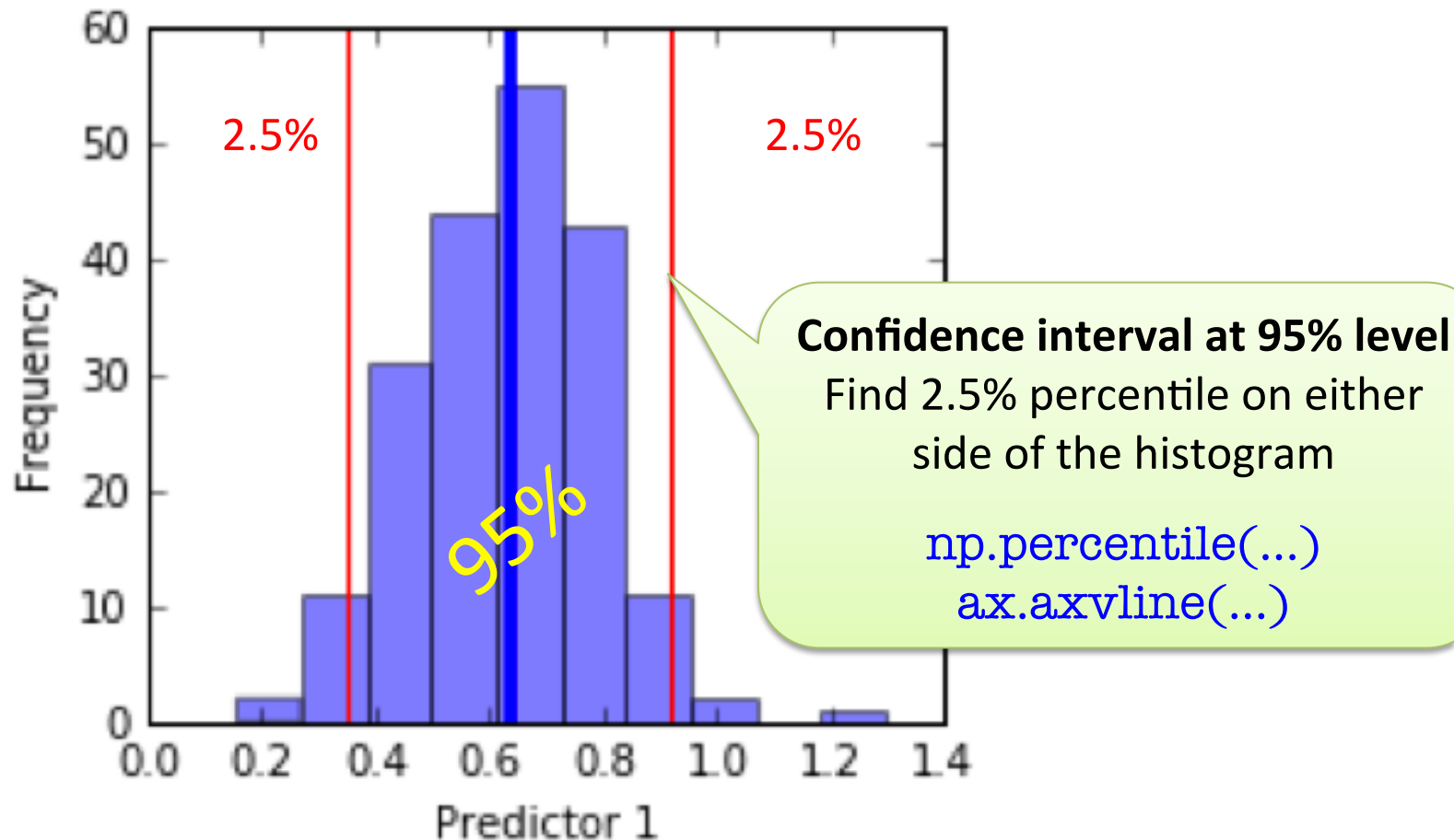
multiple_linear_regression_fit

Hint:

Use `np.random.permutation`
to permute the data set,
and pick the top 100 entries

Confidence Intervals: Subsampling

- Plot histogram of coefficients: `ax.hist(...)`



Confidence Intervals: Statsmodels

- Built-in python module

```
import statsmodels.api as sm
```

- Ordinary least squares (OLS) regression

```
# Create model for linear regression  
model = sm.OLS(y, x)
```

```
# Fit model  
fitted_model = model.fit()
```

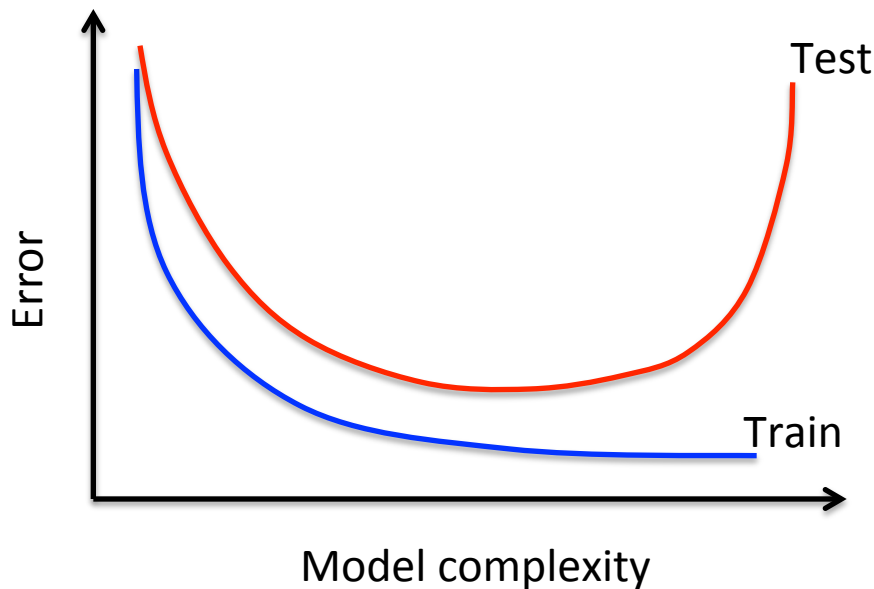
- Confidence intervals for fitted model

```
# 2D array of confidence intervals at significance level 'alpha'  
# Each row contains the confidence interval for a parameter  
conf_int = fitted_model.conf_int(alpha = 0.05)
```

Review: Model Selection

Training vs. Test errors

- Polynomial regression
 - Model complexity: Degree of polynomial
 - Is larger always better?



Model Selection Criterion

- How does one choose the 'best' polynomial degree using **only the training set**?
- Use a *model selection criterion* as a **proxy for the test error**:

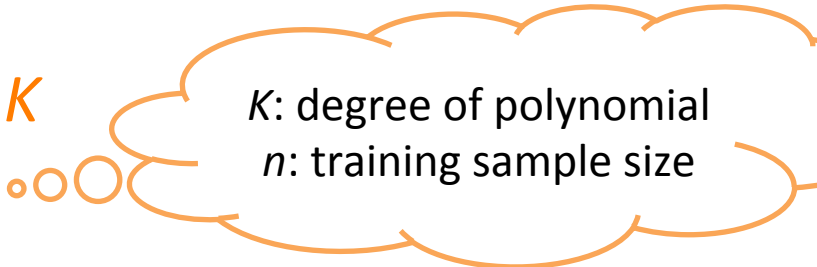
-2 x Log-likelihood + penalty term

Model Selection Criterion

- Akaike Information Criterion

- AIC = $-2 \times \text{Log-likelihood} + 2 \times K$

- For least-squares regression:



K : degree of polynomial
 n : training sample size

$$\text{AIC} = n \log \left(\frac{\text{RSS}}{n} \right) + 2K$$

- Bayesian Information Criterion (BIC)

- BIC = $-2 \times \text{Log-likelihood} + 2 \times \log(K)$

- For least-squares regression:

$$\text{BIC} = n \log \left(\frac{\text{RSS}}{n} \right) + \log(n)K$$

Note: The AIC and BIC definitions are slightly different from the text book, and correspond to the case where the residual error variance σ^2 is unknown.