

Unit 2

Multiple Linear Regression

Prof. Phil Schniter



THE OHIO STATE UNIVERSITY

ECE 5307: Introduction to Machine Learning, Sp23

Learning objectives

- Formulate a machine learning task as **multiple linear regression**
 - Understand advantage over simple linear regression
 - Identify feature and target variables
 - Recognize possibilities for **feature transformation**, such as **one-hot-coding**
- Describe the regression model in **matrix/vector** form
- Understand the **least-squares** solution for the model coefficients
 - Derive the LS solution via minimization of the RSS
 - Assess **goodness-of-fit** via R^2
 - Express the LS solution in terms of covariance matrices
- Implement linear regression in **Python** using the **Numpy** and **sklearn** packages

Outline

- Motivating Example: Predicting the Progression of Diabetes
- The Multi-Variable Linear Model
- The Least-Squares Solution
- Understanding the LS Solution
- Multiple Linear Regression in Python
- Simple vs. Multiple Linear Regression
- One-Hot Coding and Feature Transformations

Example: Predicting the progression of diabetes

- Approximately 463 million people worldwide suffer from diabetes
- The disease kills approximately 4.2 million people per year
- Can we predict the progression of the disease from biometrics like age, sex, body-mass index, blood pressure, and the results of a blood test?
- Explored in `demo02_diabetes.ipynb`



Diabetes Dataset

Data Set Characteristics:

Number of Instances:	442
Number of Attributes:	First 10 columns are numeric predictive values
Target:	Column 11 is a quantitative measure of disease progression one year after baseline
Attribute Information:	<ul style="list-style-type: none"> • age age in years • sex • bmi body mass index • bp average blood pressure • s1 tc, total serum cholesterol • s2 ldl, low-density lipoproteins • s3 hdl, high-density lipoproteins • s4 tch, total cholesterol / HDL • s5 ltg, possibly log of serum triglycerides level • s6 glu, blood sugar level

Loading the data

- Scikit-Learn (**sklearn**) package:
 - Contains many methods for machine learning
 - Contains built-in datasets too
 - We will use **sklearn** extensively!
- The Diabetes Dataset is one of sklearn's built-in datasets

```
: from sklearn import datasets, linear_model, preprocessing  
  
# Load the diabetes dataset  
diabetes = datasets.load_diabetes()  
X = diabetes.data  
y = diabetes.target
```

```
nsamp, natt = X.shape  
print("num samples={0:d} num attributes={1:d}".format(nsamp,natt))  
  
num samples=442 num attributes=10
```

Matrix/vector representation of data

- We represent the data as **feature matrix** \mathbf{X} and **target vector** \mathbf{y}
- The feature matrix \mathbf{X} is in $\mathbb{R}^{n \times d}$
 - $n = \#$ samples in dataset
 - $d = \#$ features
 - the i th row is \mathbf{x}_i^T , which contains the feature data for the i th sample
- The target vector \mathbf{y} is in $\mathbb{R}^{n \times 1}$
- The i th data sample is the pair (\mathbf{x}_i, y_i)

$$\mathbf{X} = \begin{bmatrix} x_{11} & \cdots & x_{1d} \\ \vdots & & \vdots \\ x_{n1} & \cdots & x_{nd} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix}$$

$$\mathbf{x}_i^T = [x_{i1} \quad \cdots \quad x_{id}]$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

Outline

- Motivating Example: Predicting the Progression of Diabetes
- The Multi-Variable Linear Model
- The Least-Squares Solution
- Understanding the LS Solution
- Multiple Linear Regression in Python
- Simple vs. Multiple Linear Regression
- One-Hot Coding and Feature Transformations

Multi-variable linear model

- Scalar **target** variable $y \in \mathbb{R}$
- Vector of **features** $\mathbf{x} = [x_1, \dots, x_d]^T$
 - d features, also known as predictors, attributes, or independent variables
- Linear model:

$$y \approx \beta_0 + \beta_1 x_1 + \dots + \beta_d x_d \triangleq \hat{y}$$

- \hat{y} is the linear prediction of the target y from \mathbf{x}
 - Note: a total of $d+1$ learnable coefficients in the model
- How do we choose the best prediction coefficients $\boldsymbol{\beta} = [\beta_0, \dots, \beta_d]^T$ given the data $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$?

Linear regression using vectors & matrices

- The predicted target for the i th sample is

$$\hat{y}_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_d x_{id}$$

- Let's define the **augmented feature matrix** \mathbf{A} and the **coefficient vector** $\boldsymbol{\beta}$:

$$\mathbf{A} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1d} \\ \vdots & \vdots & & \vdots \\ 1 & x_{n1} & \cdots & x_{nd} \end{bmatrix}, \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \vdots \\ \beta_d \end{bmatrix}$$

- Then the vector of predicted targets $\hat{\mathbf{y}} = [\hat{y}_1, \dots, \hat{y}_n]^\top$ is

$$\hat{\mathbf{y}} = \mathbf{A}\boldsymbol{\beta}$$

- And, given a new feature vector \mathbf{x} , the predicted target would be

$$\hat{y}(\mathbf{x}) = [1 \ \mathbf{x}^\top] \boldsymbol{\beta}$$

Outline

- Motivating Example: Predicting the Progression of Diabetes
- The Multi-Variable Linear Model
- The Least-Squares Solution
- Understanding the LS Solution
- Multiple Linear Regression in Python
- Simple vs. Multiple Linear Regression
- One-Hot Coding and Feature Transformations

The least-squares problem

- We select the parameters $\beta = [\beta_0, \dots, \beta_d]^T$ of our linear model

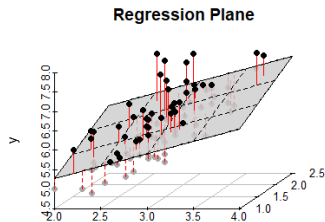
$$\hat{y}_i = \beta_0 + \beta_1 x_{i1} + \dots + \beta_d x_{id}$$

as the **least-squares fit** to the data $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$

- In other words, we choose β to minimize the **residual sum of squares** (RSS):

$$\text{RSS}(\beta) \triangleq \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- Also called the **sum of squared errors** (SSE) and **sum of square residuals** (SSR)
- Note that \hat{y}_i is implicitly a function of β
- This finds the regression plane that minimizes the sum-squared vertical deviations in the figure



RSS as a squared norm

- Recall that the RSS is defined as

$$\text{RSS}(\beta) = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

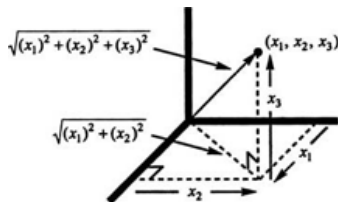
- Let us define the **norm** of a real vector x as

$$\|x\| = \sqrt{\sum_j x_j^2}$$

- A norm measures “distance” from the origin
- We use the standard **Euclidean norm**, or ℓ_2 norm
- This allows us to write the RSS as

$$\text{RSS}(\beta) = \|y - \hat{y}\|^2 = \|y - A\beta\|^2$$

- A commonly used way to express RSS!



The optimization approach: A general ML recipe

General ML problem

- Assume a **model** with some **parameters**
- Get **training data**
- Choose a **loss function**
- Find parameters that **minimize** loss

Multiple Linear Regression

- Linear model: $\hat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_d x_d$
- Data: $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$
- $\text{RSS}(\boldsymbol{\beta}) \triangleq \sum_{i=1}^n (y_i - \hat{y}_i)^2$
- Find $\boldsymbol{\beta} = [\beta_0, \cdots, \beta_d]^\top$ that minimizes $\text{RSS}(\boldsymbol{\beta})$

Minimizing a quadratic function

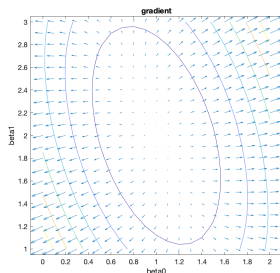
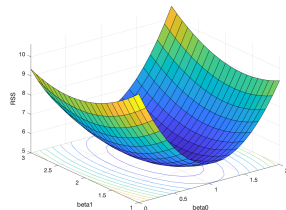
- RSS is a quadratic function:

$$\begin{aligned}\text{RSS}(\beta) &= \|\mathbf{y} - \mathbf{A}\beta\|^2 = (\mathbf{y} - \mathbf{A}\beta)^\top (\mathbf{y} - \mathbf{A}\beta) \\ &= \mathbf{y}^\top \mathbf{y} - 2\mathbf{y}^\top \mathbf{A}\beta + \beta^\top \mathbf{A}^\top \mathbf{A}\beta\end{aligned}$$

- Consider the gradient and Hessian:

$$\begin{bmatrix} \vdots \\ \frac{\partial \text{RSS}(\beta)}{\partial \beta_j} \\ \vdots \end{bmatrix}, \quad \begin{bmatrix} \vdots & & \\ \cdots & \frac{\partial^2 \text{RSS}(\beta)}{\partial \beta_j \partial \beta_k} & \cdots \\ \vdots & & \end{bmatrix}$$

- For a quadratic function with a positive semi-definite (PSD) Hessian, any β that zeros the gradient is a minimum
- For RSS, can show Hessian is PSD (i.e., function curves upward) everywhere



The least-squares solution

- Writing the RSS and target prediction as

$$\text{RSS}(\boldsymbol{\beta}) = \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad \text{with} \quad \hat{y}_i = \sum_{j=0}^d a_{ij} \beta_j \quad \text{where} \quad a_{ij} = [\mathbf{A}]_{ij},$$

we can use the chain rule to obtain

$$\frac{\partial \text{RSS}(\boldsymbol{\beta})}{\partial \beta_j} = -2 \sum_{i=1}^n (y_i - \hat{y}_i) a_{ij} = -2 [\mathbf{A}^\top (\mathbf{y} - \hat{\mathbf{y}})]_j = -2 [\mathbf{A}^\top (\mathbf{y} - \mathbf{A}\boldsymbol{\beta})]_j$$

- Stacking these into the gradient vector $\nabla \text{RSS}(\boldsymbol{\beta})$ and setting it to zero gives

$$\begin{aligned} \mathbf{0} &= \mathbf{A}^\top (\mathbf{y} - \mathbf{A}\boldsymbol{\beta}_{\text{ls}}) \\ \Leftrightarrow \mathbf{A}^\top \mathbf{A} \boldsymbol{\beta}_{\text{ls}} &= \mathbf{A}^\top \mathbf{y} \\ \Leftrightarrow \boxed{\boldsymbol{\beta}_{\text{ls}} = (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top \mathbf{y}} &\quad \text{assuming } \mathbf{A}^\top \mathbf{A} \text{ is invertible} \end{aligned}$$

- Note: if $n < d$ then $\mathbf{A}^\top \mathbf{A}$ isn't invertible. We'll talk about this later.

R^2 Goodness-of-fit

- Question: How to judge whether a predictor is doing “well” on the dataset?
- Answer: Use a *normalized* version of RSS:
 - RSS includes contributions from n training samples.
By considering RSS/n , we remove the dependence on n .
 - RSS/n depends on s_y^2 , the variance-of- y (i.e., if s_y^2 doubles then RSS/n doubles).
By considering $\frac{\text{RSS}/n}{s_y^2}$, we remove the dependence on s_y^2 .
- More commonly, we report

$$1 - \frac{\text{RSS}/n}{s_y^2} \triangleq R^2,$$

known as the “coefficient of determination”

https://en.wikipedia.org/wiki/Coefficient_of_determination

- $R^2 = 1$ implies that the predictor is perfect (i.e., $\hat{y}_i = y_i$)
- $R^2 = 0$ implies the predictor is no better than the trivial one (i.e., $\hat{y}_i = \bar{y}$)
- $R^2 < 0$ implies worse than trivial!

Outline

- Motivating Example: Predicting the Progression of Diabetes
- The Multi-Variable Linear Model
- The Least-Squares Solution
- Understanding the LS Solution
- Multiple Linear Regression in Python
- Simple vs. Multiple Linear Regression
- One-Hot Coding and Feature Transformations

Understanding the LS solution

- We derived the expression

$$\beta_{ls} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{y}$$

for the RSS-minimizing version of the prediction coefficients β using

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1d} \\ \vdots & \vdots & & \vdots \\ 1 & x_{n1} & \cdots & x_{nd} \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_0 \\ \vdots \\ \beta_d \end{bmatrix}$$

- But the above expression is not very insightful
- Can we express β_{ls} using sample statistics of the data $\{(x_i, y_i)\}_{i=1}^n$?

Slopes and intercept

- Recall the linear prediction equation

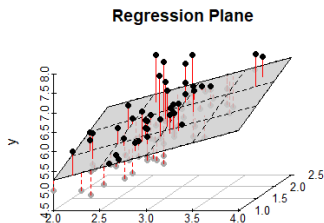
$$\hat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_d x_d$$

- Let's partition the coefficients into first-and-others, i.e., $\beta^T = [\beta_0 \ \beta_{1:d}^T]$
 - As before, β_0 is the intercept
 - $\beta_{1:d} = [\beta_1, \dots, \beta_d]^T$ contains slope coefficients

- With this notation, we can write

$$\hat{y} = \beta_0 + \beta_{1:d}^T x$$

which will be convenient later.



Sample auto-covariance and cross-covariance

- The sample statistics that we'll need are

$$\bar{\mathbf{x}} \triangleq \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i = \begin{bmatrix} \bar{x}_1 \\ \vdots \\ \bar{x}_d \end{bmatrix}, \quad \bar{y} \triangleq \frac{1}{n} \sum_{i=1}^n y_i \quad \text{sample means}$$

$$\mathbf{S}_{xx} \triangleq \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T \quad \text{sample auto-covariance matrix}$$

$$\mathbf{s}_{xy} \triangleq \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i - \bar{\mathbf{x}})(y_i - \bar{y}) \quad \text{sample cross-covariance vector}$$

- These are matrix/vector extensions of the scalar quantities that we saw in simple linear regression:

$$s_{xx} = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})(x_i - \bar{x}), \quad s_{xy} = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$$

Alternative expressions for auto- & cross-covariance

- We can also express the auto-covariance as follows:

$$\begin{aligned}
 S_{xx} &= \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T \\
 &= \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i \mathbf{x}_i^T - \underbrace{\bar{\mathbf{x}} \left(\frac{1}{n} \sum_{i=1}^n \mathbf{x}_i \right)^T}_{=\bar{\mathbf{x}}} - \underbrace{\left(\frac{1}{n} \sum_{i=1}^n \mathbf{x}_i \right) \bar{\mathbf{x}}^T}_{=\bar{\mathbf{x}}} + \bar{\mathbf{x}} \bar{\mathbf{x}}^T = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i \mathbf{x}_i^T - \bar{\mathbf{x}} \bar{\mathbf{x}}^T
 \end{aligned}$$

- Similarly, we can express the cross-covariance as follows:

$$\begin{aligned}
 s_{xy} &= \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i - \bar{\mathbf{x}})(y_i - \bar{y}) \\
 &= \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i y_i - \underbrace{\bar{\mathbf{x}} \left(\frac{1}{n} \sum_{i=1}^n y_i \right)}_{=\bar{y}} - \underbrace{\left(\frac{1}{n} \sum_{i=1}^n \mathbf{x}_i \right) \bar{y}}_{=\bar{\mathbf{x}}} + \bar{\mathbf{x}} \bar{y} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i y_i - \bar{\mathbf{x}} \bar{y}
 \end{aligned}$$

Minimizing RSS: Derivation

The minimum RSS is achieved by values of $(\beta_0, \dots, \beta_d)$ that zero the gradient, i.e.,

$$0 = \frac{\partial \text{RSS}(\beta_0, \beta_1)}{\partial \beta_0} = \frac{\partial}{\partial \beta_0} \sum_{i=1}^n (y_i - \beta_0 - \mathbf{x}_i^\top \boldsymbol{\beta}_{1:d})^2 = -2 \sum_{i=1}^n (y_i - \beta_0 - \mathbf{x}_i^\top \boldsymbol{\beta}_{1:d}) \quad (1)$$

and, for all $j = 1, \dots, d$:

$$0 = \frac{\partial \text{RSS}(\beta_0, \beta_j)}{\partial \beta_j} = \frac{\partial}{\partial \beta_j} \sum_{i=1}^n (y_i - \beta_0 - \underbrace{\mathbf{x}_i^\top \boldsymbol{\beta}_{1:d}}_{\sum_{j'=1}^d x_{ij'} \beta_{j'}})^2 = -2 \sum_{i=1}^n x_{ij} (y_i - \beta_0 - \mathbf{x}_i^\top \boldsymbol{\beta}_{1:d}) \quad (2)$$

Starting with (1), we can multiply both sides by $-\frac{1}{2n}$ to give

$$0 = \frac{1}{n} \sum_{i=1}^n (y_i - \beta_0 - \mathbf{x}_i^\top \boldsymbol{\beta}_{1:d}) = \underbrace{\frac{1}{n} \sum_{i=1}^n y_i}_{\bar{y}} - \beta_0 - \underbrace{\frac{1}{n} \sum_{i=1}^n \mathbf{x}_i^\top}_{\bar{\mathbf{x}}^\top} \boldsymbol{\beta}_{1:d}$$

$$\Leftrightarrow \boxed{\beta_0 = \bar{y} - \bar{\mathbf{x}}^\top \boldsymbol{\beta}_{1:d}} \quad (3)$$

Minimizing RSS: Derivation (cont.)

Similarly, we can multiply both sides of (2) by $-\frac{1}{2n}$ to give

$$0 = \frac{1}{n} \sum_{i=1}^n x_{ij}(y_i - \beta_0 - \mathbf{x}_i^\top \boldsymbol{\beta}_{1:d}) = \frac{1}{n} \sum_{i=1}^n x_{ij}y_i - \bar{x}_j\beta_0 - \left(\frac{1}{n} \sum_{i=1}^n x_{ij}\mathbf{x}_i^\top\right)\boldsymbol{\beta}_{1:d}$$

and plug in (3) to get

$$\begin{aligned} 0 &= \frac{1}{n} \sum_{i=1}^n x_{ij}y_i - \bar{x}_j\bar{y} + \bar{x}_j\bar{\mathbf{x}}^\top \boldsymbol{\beta}_{1:d} - \left(\frac{1}{n} \sum_{i=1}^n x_{ij}\mathbf{x}_i^\top\right)\boldsymbol{\beta}_{1:d} \\ &= \left[\frac{1}{n} \sum_{i=1}^n \mathbf{x}_i y_i - \bar{\mathbf{x}}\bar{y} + \bar{\mathbf{x}}\bar{\mathbf{x}}^\top \boldsymbol{\beta}_{1:d} - \left(\frac{1}{n} \sum_{i=1}^n \mathbf{x}_i \mathbf{x}_i^\top\right)\boldsymbol{\beta}_{1:d} \right]_j \\ &= [\mathbf{s}_{xy} - \mathbf{S}_{xx}\boldsymbol{\beta}_{1:d}]_j \end{aligned} \tag{4}$$

using the alternate expressions for \mathbf{S}_{xx} and \mathbf{s}_{xy} .

Since (4) must hold at all $j = 1, \dots, d$, we have

$$\mathbf{0} = \mathbf{s}_{xy} - \mathbf{S}_{xx}\boldsymbol{\beta}_{1:d} \quad \Leftrightarrow \quad \boxed{\boldsymbol{\beta}_{1:d} = \mathbf{S}_{xx}^{-1}\mathbf{s}_{xy}} \quad \text{assuming } \mathbf{S}_{xx} \text{ is invertible}$$

The LS solution via sample covariance statistics

- We can summarize the LS solution as

$$\boldsymbol{\beta}_{\text{ls}} = \begin{bmatrix} \beta_0 \\ \boldsymbol{\beta}_{1:d} \end{bmatrix} = \begin{bmatrix} \bar{y} - \bar{\mathbf{x}}^T \mathbf{S}_{xx}^{-1} \mathbf{s}_{xy} \\ \mathbf{S}_{xx}^{-1} \mathbf{s}_{xy} \end{bmatrix}$$

- In the special case that $d = 1$ (i.e., simple linear regression), we have

$$\begin{array}{l} \mathbf{x}_i \rightarrow x_i \\ \bar{\mathbf{x}} \rightarrow \bar{x} \\ \mathbf{S}_{xx} \rightarrow s_{xx} \\ \mathbf{s}_{xy} \rightarrow s_{xy} \end{array} \quad \Rightarrow \quad \boldsymbol{\beta}_{\text{ls}} = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} \bar{y} - \bar{x}s_{xy}/s_{xx} \\ s_{xy}/s_{xx} \end{bmatrix}$$

which are the same expressions that we derived in the previous lecture packet.

Outline

- Motivating Example: Predicting the Progression of Diabetes
- The Multi-Variable Linear Model
- The Least-Squares Solution
- Understanding the LS Solution
- Multiple Linear Regression in Python
- Simple vs. Multiple Linear Regression
- One-Hot Coding and Feature Transformations

Manually computing the LS solution with Numpy

- We could use Numpy routines to solve for the LS solution

$$\beta_{ls} = (A^T A)^{-1} A^T y \text{ with } A = [1 \ X]$$

```
ones = np.ones((nsamp,1))  
A = np.hstack((ones,X))
```

but explicitly computing the matrix inverse is very slow for large matrices

- It's much more efficient to attack the LS problem “ $\arg \min_{\beta} \|A\beta - y\|^2$ ” directly via

```
out = np.linalg.lstsq(A,y)  
beta = out[0]
```

since `np.linalg.lstsq` uses more numerically efficient LAPACK routines.

Linear regression via sklearn

- An even better way to implement linear regression is with `sklearn`. There, we create a `LinearRegression` object and then call its `fit` method to design the LS coefficients.
- In the diabetes demo, we design the linear predictor from the training data, and then apply it to the test data using the `predict` method. This gives $R^2 = 0.51$.

```
regr = linear_model.LinearRegression()
regr.fit(X,y)
```

```
y_pred = regr.predict(X)
RSS = np.sum((y_pred-y)**2)
Rsqr = 1-RSS/nsamp/(np.std(y)**2)
print("R^2 = {0:f}".format(Rsqr))
```

```
R^2 = 0.514031
```

Outline

- Motivating Example: Predicting the Progression of Diabetes
- The Multi-Variable Linear Model
- The Least-Squares Solution
- Understanding the LS Solution
- Multiple Linear Regression in Python
- Simple vs. Multiple Linear Regression
- One-Hot Coding and Feature Transformations

Simple versus multiple linear regression

Recall...

- Simple linear regression: **one** feature/predictor
 - scalar feature x
 - linear model: $\hat{y} \approx \beta_0 + \beta_1 x$
- Multiple linear regression: **multiple** features/predictors
 - feature vector $\mathbf{x} = [x_1, \dots, x_d]^T$
 - linear model: $\hat{y} \approx \beta_0 + \beta_1 x_1 + \dots + \beta_d x_d$
 - reduces to simple linear regression when $d = 1$

So...

- Why use multiple linear regression?
 - Will it always improve on simple linear regression?
 - When will it give the same result?

Simple linear regression for the diabetes demo

- Idea: Fit each feature x_j individually
- How well does this work? Compute the R_j^2 coefficient for each feature j
 - The best predictor gives $R_j^2 = 0.34$
- Recall that for multiple linear regression, we got $R^2 = 0.51$.
 - Thus multiple linear regression outperforms simple linear regression on this dataset
 - In generally, LS linear regression performs at least as well as LS simple regression

```
ym = np.mean(y)
syy = np.mean((y-ym)**2)
Rsq = np.zeros(natt)
beta0 = np.zeros(natt)
beta1 = np.zeros(natt)
for j in range(natt):
    xm = np.mean(X[:,j])
    sxy = np.mean((X[:,j]-xm)*(y-ym))
    sxx = np.mean((X[:,j]-xm)**2)
    beta1[j] = sxy/sxx
    beta0[j] = ym - beta1[j]*xm
    Rsq[j] = (sxy)**2/sxx/syy

print("j={0:1d} R^2={1:f} beta0={2:f} "
```

```
j=0 R^2=0.035302 beta0=152.133484 beta1=30
j=1 R^2=0.001854 beta0=152.133484 beta1=69
j=2 R^2=0.343924 beta0=152.133484 beta1=94
j=3 R^2=0.194908 beta0=152.133484 beta1=71
j=4 R^2=0.044954 beta0=152.133484 beta1=34
j=5 R^2=0.030295 beta0=152.133484 beta1=28
j=6 R^2=0.155859 beta0=152.133484 beta1=-6
j=7 R^2=0.185290 beta0=152.133484 beta1=69
j=8 R^2=0.320224 beta0=152.133484 beta1=91
j=9 R^2=0.146294 beta0=152.133484 beta1=61
```

Partitioning into training & testing subsets

- In practice, we design β to predict the target variables of *unlabeled* data x
 - Predicting the target variables of labeled data (x, y) is trivial; we know them!
- To mimic this situation, we partition our diabetes dataset into two subsets:
 - **Training data:** First 300 samples
 - **Test data:** Remaining 142 samples

```
ns_train = 300
ns_test = nsamp - ns_train
X_tr = X[:ns_train,:]
y_tr = y[:ns_train]
X_test = X[ns_train:,:]
y_test = y[ns_train:]
```

Then we design β using the training data, and evaluate performance (e.g., RSS) on the test data:

```
regr.fit(X_tr,y_tr)
y_tr_pred = regr.predict(X_tr)
y_test_pred = regr.predict(X_test)
```

```
R^2_tr   =      0.514719
R^2_test =      0.507199
```

As expected, the training predictions are slightly better than the test predictions

- We will discuss train/test splits in much more detail in the next unit

Outline

- Motivating Example: Predicting the Progression of Diabetes
- The Multi-Variable Linear Model
- The Least-Squares Solution
- Understanding the LS Solution
- Multiple Linear Regression in Python
- Simple vs. Multiple Linear Regression
- One-Hot Coding and Feature Transformations

One-hot coding

- Suppose some features are **categorical** variables
 - Ex: We want to predict the mpg y of a car, given its horsepower x_1 and brand x_2 , where the brands are $\{\text{Ford}, \text{Honda}, \text{BMW}\}$.
 - Problem: Ordinal numbers like $\{1, 2, 3\}$ are not appropriate for brands. Why?
 - Solution: “**One-hot coding**”: Code brands as *binary vectors*!

- Example of **one-hot coding**:

Brand	$x_2^{(1)}$	$x_2^{(2)}$	$x_2^{(3)}$
Ford	1	0	0
Honda	0	1	0
BMW	0	0	1

- Since x_2 has 3 possible categories, represent it using:
 - Linear model becomes $y \approx \beta_0 + \beta_1 x_1 + \beta_2 x_2^{(1)} + \beta_3 x_2^{(2)} + \beta_4 x_2^{(3)}$
 - Essentially, this gives 3 *different* linear models with same slope:
 - Ford: $y \approx \beta_0 + \beta_1 x_1 + \beta_2$
 - Honda: $y \approx \beta_0 + \beta_1 x_1 + \beta_3$
 - BMW: $y \approx \beta_0 + \beta_1 x_1 + \beta_4$
 - Interpretation: **One-hot coding** allows a different intercept for each category!

One-hot coding of slope

- Previously, we saw a form of one-hot coding that led to category-dependent intercepts
- We could use a similar approach to get category-dependent slopes
- Example:
 - Ex: We want to predict the mpg y of a car, given its horsepower x_1 and brand x_2 , where the brands are {Ford,Honda,BMW}. But we suspect that different brands use different methods to measure horsepower x_1 .

Brand	$x_2^{(1)}$	$x_2^{(2)}$	$x_2^{(3)}$
Ford	1	0	0
Honda	0	1	0
BMW	0	0	1

- Since x_2 has 3 possible categories, represent it using:
 - Adopt the model $y \approx \beta_0 + \beta_1 x_1 x_2^{(1)} + \beta_2 x_1 x_2^{(2)} + \beta_3 x_1 x_2^{(3)}$
 - Essentially, this gives 3 *different* linear models with same intercept:
 - Ford: $y \approx \beta_0 + \beta_1 x_1$
 - Honda: $y \approx \beta_0 + \beta_2 x_1$
 - BMW: $y \approx \beta_0 + \beta_3 x_1$
 - Interpretation: One-hot coding allows a different slope for each category!

Invertibility issues due to one-hot coding

- Be careful: One-hot coding the intercept can make $\mathbf{A}^\top \mathbf{A}$ non-invertible!
- For example, recall the case where x_2 was one of 3 categories, and we used

Brand	$x_2^{(1)}$	$x_2^{(2)}$	$x_2^{(3)}$
Ford	1	0	0
Honda	0	1	0
BMW	0	0	1

to give $y \approx \beta_0 + \beta_1 x_1 + \beta_2 x_2^{(1)} + \beta_3 x_2^{(2)} + \beta_4 x_2^{(3)}$

- In this case, $\beta = [\beta_0, \beta_1, \beta_2, \beta_3, \beta_4]^\top$ and the rows of \mathbf{A} will all have the form

$$[1, x_{i1}, 1, 0, 0] \text{ or } [1, x_{i1}, 0, 1, 0] \text{ or } [1, x_{i1}, 0, 0, 1].$$

- Note that $\beta = [-1, 0, 1, 1, 1]^\top$ yields $\mathbf{A}\beta = \mathbf{0}$, as well as $\mathbf{A}^\top \mathbf{A}\beta = \mathbf{0}$
- Because there exists $\beta \neq \mathbf{0}$ such that $\mathbf{A}^\top \mathbf{A}\beta = \mathbf{0}$, we know $\mathbf{A}^\top \mathbf{A}$ is **singular**

- To circumvent this problem, ...

- Use a non-redundant one-hot coding scheme, e.g.,
- Use **regularization** (see Unit 4)

Brand	$x_2^{(1)}$	$x_2^{(2)}$
Ford	1	0
Honda	0	1
BMW	0	0

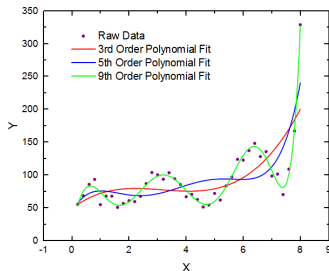
Polynomial regression

- Suppose that y depends only on a single variable x , and we want to model y as a **polynomial function** of x :

$$y \approx \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_d x^d$$

since this may perform better than linear regression

- Easy to handle using multiple linear regression:
Simply define $x_j \triangleq x^j$ for $j = 1, \dots, d$
- Note: same idea can be used for other nonlinear models, e.g., $x_j \triangleq \cos(\omega_j x)$!
- Like one-hot coding, this is an instance of **feature transformation**
- Problem: how do we choose the polynomial order d ?
 - Will discuss this in Unit 3



Learning objectives

- Formulate a machine learning task as **multiple linear regression**
 - Understand advantage over simple linear regression
 - Identify feature and target variables
 - Recognize possibilities for **feature transformation**, such as **one-hot-coding**
- Describe the regression model in **matrix/vector** form
- Understand the **least-squares** solution for the model coefficients
 - Derive the LS solution via minimization of the RSS
 - Assess **goodness-of-fit** via R^2
 - Express the LS solution in terms of correlation and covariance matrices
- Implement linear regression in **Python** using the **Numpy** and **sklearn** packages