Unit 2
Multiple Linear Regression

Prof. Phil Schniter

THE OHIO STATE UNIVERSITY

ECE 5307: Introduction to Machine Learning, Sp23

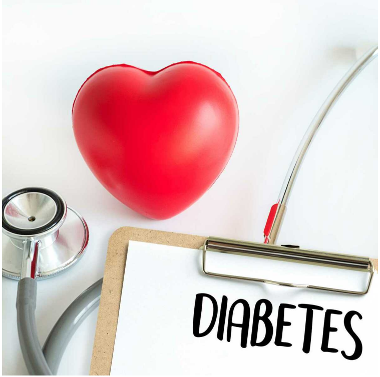
Learning objectives

- 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

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:	• 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 $\boldsymbol{X}$ and target vector $\boldsymbol{y}$
- The feature matrix $\boldsymbol{X}$ is in $\mathbb{R}^{n \times d}$
  - $n = \#$ samples in dataset
  - $d = \#$ features
  - the ith row is $\boldsymbol{x}_i^\mathsf{T}$, which contains the feature data for the ith sample
- The target vector $\boldsymbol{y}$ is in $\mathbb{R}^{n \times 1}$
- The ith data sample is the pair $(\boldsymbol{x}_i, y_i)$

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

$$\boldsymbol{x}_i^\mathsf{T} = \begin{bmatrix} x_{i1} & \cdots & x_{id} \end{bmatrix}$$

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

Outline

Multi-variable linear model

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

$$y \approx \beta_0 + \beta_1 x_1 + \cdots + \beta_d x_d \triangleq \widehat{y}$$

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

Linear regression using vectors & matrices

- The predicted target for the ith sample is

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

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

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

- Then the vector of predicted targets $\widehat{\boldsymbol{y}} = [\widehat{y}_1, \ldots, \widehat{y}_n]^\mathsf{T}$ is

$$\widehat{\boldsymbol{y}} = \boldsymbol{A}\boldsymbol{\beta}$$

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

$$\widehat{y}(\boldsymbol{x}) = [1 \;\; \boldsymbol{x}^\mathsf{T}]\boldsymbol{\beta}$$

Outline

The least-squares problem

- We select the parameters $\boldsymbol{\beta} = [\beta_0, \ldots, \beta_d]^{\mathsf{T}}$ of our linear model

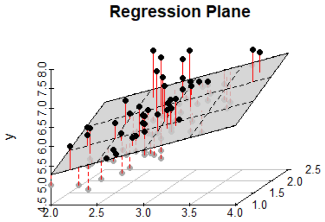
$$\widehat{y}_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_d x_{id}$$

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

- In other words, we choose $\boldsymbol{\beta}$ to minimize the residual sum of squares (RSS):

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

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



RSS as a squared norm

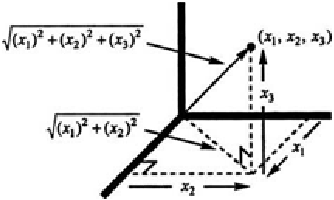
- Recall that the RSS is defined as

$$\mathrm{RSS}(\boldsymbol{\beta}) = \sum_{i=1}^{n} (y_i - \widehat{y}_i)^2$$

- Let us define the norm of a real vector $\boldsymbol{x}$ as

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

  - 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

$$\mathrm{RSS}(\boldsymbol{\beta}) = \|\boldsymbol{y} - \widehat{\boldsymbol{y}}\|^2 = \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{\beta}\|^2$$

- A commonly used way to express RSS!

The optimization approach: A general ML recipe

General ML problem		Multiple Linear Regression
Assume a model with some parameters	$\rightarrow$	Linear model: $\widehat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_d x_d$
Get training data	$\rightarrow$	Data: $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$
Choose a loss function	$\rightarrow$	$\mathrm{RSS}(\boldsymbol{\beta}) \triangleq \sum_{i=1}^n (y_i - \widehat{y}_i)^2$
Find parameters that minimize loss	$\rightarrow$	Find $\boldsymbol{\beta} = [\beta_0, \cdots, \beta_d]^\mathsf{T}$ that minimizes $\mathrm{RSS}(\boldsymbol{\beta})$

Minimizing a quadratic function

- RSS is a quadratic function:

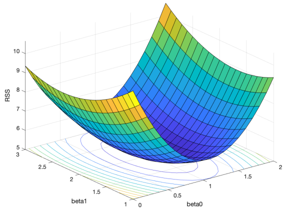
$$\begin{aligned}\mathrm{RSS}(\boldsymbol{\beta}) &= \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{\beta}\|^2 = (\boldsymbol{y} - \boldsymbol{A}\boldsymbol{\beta})^{\mathsf{T}}(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{\beta}) \\ &= \boldsymbol{y}^{\mathsf{T}}\boldsymbol{y} - 2\boldsymbol{y}^{\mathsf{T}}\boldsymbol{A}\boldsymbol{\beta} + \boldsymbol{\beta}^{\mathsf{T}}\boldsymbol{A}^{\mathsf{T}}\boldsymbol{A}\boldsymbol{\beta}\end{aligned}$$

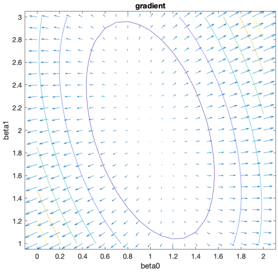
- Consider the gradient and Hessian:

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

- For a quadratic function with a positive semi-definite (PSD) Hessian, any $\boldsymbol{\beta}$ that zeros the gradient is a minimum

- For RSS, can show Hessian is PSD (i.e., function curves upward) everwhere





The least-squares solution

- Writing the RSS and target prediction as

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

we can use the chain rule to obtain

$$\frac{\partial\,\mathrm{RSS}(\boldsymbol{\beta})}{\partial \beta_j} = -2\sum_{i=1}^{n} (y_i - \widehat{y}_i) a_{ij} = -2\big[\boldsymbol{A}^{\mathsf{T}}(\boldsymbol{y} - \widehat{\boldsymbol{y}})\big]_j = -2\big[\boldsymbol{A}^{\mathsf{T}}(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{\beta})\big]_j$$

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

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

  - Note: if $n < d$ then $\boldsymbol{A}^{\mathsf{T}}\boldsymbol{A}$ 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 $\frac{\mathrm{RSS}/n}{s_y^2}$, we remove the dependence on s_y^2.
- More commonly, we report

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

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

Outline

Understanding the LS solution

- We derived the expression

$$\boldsymbol{\beta}_{\mathsf{ls}} = (\boldsymbol{A}^{\mathsf{T}}\boldsymbol{A})^{-1}\boldsymbol{A}^{\mathsf{T}}\boldsymbol{y}$$

for the RSS-minimizing version of the prediction coefficients $\boldsymbol{\beta}$ using

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

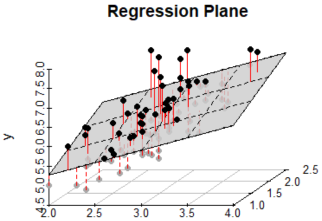
- But the above expression is not very insightful
- Can we express $\boldsymbol{\beta}_{\mathsf{ls}}$ using sample statistics of the data $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$?

Slopes and intercept

- Recall the linear prediction equation

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

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



- With this notation, we can write

$$\widehat{y} = \color{blue}{\beta_0 + \boldsymbol{\beta}_{1:d}^\mathsf{T} \boldsymbol{x}}$$

which will be convenient later.

Sample auto-covariance and cross-covariance

- The sample statistics that we'll need are

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

$$\boldsymbol{S}_{xx} \triangleq \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i-\overline{\boldsymbol{x}})(\boldsymbol{x}_i-\overline{\boldsymbol{x}})^{\mathsf{T}} \qquad \text{sample auto-covariance matrix}$$

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

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

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

Alternative expressions for auto- & cross-covariance

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

$$\boldsymbol{S}_{xx} = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i - \overline{\boldsymbol{x}})(\boldsymbol{x}_i - \overline{\boldsymbol{x}})^\mathsf{T}$$

$$= \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}_i\boldsymbol{x}_i^\mathsf{T} - \overline{\boldsymbol{x}}\underbrace{\Big(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}_i\Big)^\mathsf{T}}_{=\overline{\boldsymbol{x}}} - \underbrace{\Big(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}_i\Big)}_{=\overline{\boldsymbol{x}}}\overline{\boldsymbol{x}}^\mathsf{T} + \overline{\boldsymbol{x}}\,\overline{\boldsymbol{x}}^\mathsf{T} = \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}_i\boldsymbol{x}_i^\mathsf{T} - \overline{\boldsymbol{x}}\,\overline{\boldsymbol{x}}^\mathsf{T}$$

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

$$\boldsymbol{s}_{xy} = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i - \overline{\boldsymbol{x}})(y_i - \overline{y})$$

$$= \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}_i y_i - \overline{\boldsymbol{x}}\underbrace{\Big(\frac{1}{n}\sum_{i=1}^{n}y_i\Big)}_{=\overline{y}} - \underbrace{\Big(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}_i\Big)}_{=\overline{\boldsymbol{x}}}\overline{y} + \overline{\boldsymbol{x}}\,\overline{y} = \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}_i y_i - \overline{\boldsymbol{x}}\,\overline{y}$$

Minimizing RSS: Derivation

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

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

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

$$0 = \frac{\partial\, \mathrm{RSS}(\beta_0, \beta_j)}{\partial \beta_j} = \frac{\partial}{\partial \beta_j} \sum_{i=1}^{n} (y_i - \beta_0 - \underbrace{\boldsymbol{x}_i^{\mathsf{T}} \boldsymbol{\beta}_{1:d}}_{\sum_{j'=1}^{d} x_{ij'} \beta_{j'}})^2 = -2 \sum_{i=1}^{n} x_{ij} (y_i - \beta_0 - \boldsymbol{x}_i^{\mathsf{T}} \boldsymbol{\beta}_{1:d}) \quad (2)$$

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

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

$$\Leftrightarrow \quad \boxed{\beta_0 = \overline{y} - \overline{\boldsymbol{x}}^{\mathsf{T}} \boldsymbol{\beta}_{1:d}} \quad (3)$$

Minimizing RSS: Derivation (cont.)

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

$$0 = \frac{1}{n}\sum_{i=1}^{n} x_{ij}(y_i - \beta_0 - \boldsymbol{x}_i^{\mathsf{T}}\boldsymbol{\beta}_{1:d}) = \frac{1}{n}\sum_{i=1}^{n} x_{ij}y_i - \overline{x}_j\beta_0 - \Big(\frac{1}{n}\sum_{i=1}^{n} x_{ij}\boldsymbol{x}_i^{\mathsf{T}}\Big)\boldsymbol{\beta}_{1:d}$$

and plug in (3) to get

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

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

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

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

assuming $\boldsymbol{S}_{xx}$ is invertible

The LS solution via sample covariance statistics

- We can summarize the LS solution as

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

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

$$\begin{aligned} \boldsymbol{x}_i &\to x_i \\ \overline{\boldsymbol{x}} &\to \overline{x} \\ \boldsymbol{S}_{xx} &\to s_{xx} \\ \boldsymbol{s}_{xy} &\to s_{xy} \end{aligned} \quad \Rightarrow \quad \boldsymbol{\beta}_{\mathsf{ls}} = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} \overline{y} - \overline{x} s_{xy} / s_{xx} \\ s_{xy} / s_{xx} \end{bmatrix}$$

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

Outline

Manually computing the LS solution with Numpy

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

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

```
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 "$\arg\min_{\boldsymbol{\beta}} \|\boldsymbol{A}\boldsymbol{\beta} - \boldsymbol{y}\|^2$" 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.

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

- 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 $R^2 = 0.51$.

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

```
R^2    =              0.514031
```

Outline

Simple versus multiple linear regression

Recall...

- Simple linear regression: one feature/predictor
  - scalar feature x
  - linear model: $\widehat{y} \approx \beta_0 + \beta_1 x$
- Multiple linear regression: multiple features/predictors
  - feature vector $\boldsymbol{x} = [x_1, \ldots, x_d]^\mathsf{T}$
  - linear model: $\widehat{y} \approx \beta_0 + \beta_1 x_1 + \cdots + \beta_d x_d$
  - 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 $\boldsymbol{\beta}$ to predict the target variables of *unlabeled* data $\boldsymbol{x}$
  - Predicting the target variables of labeled data $(\boldsymbol{x}, y)$ 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 $\boldsymbol{\beta}$ 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

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 {Ford,Honda,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:
  - Since x_2 has 3 possible categories, represent it using:

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

  - 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.
  - Since x_2 has 3 possible categories, represent it using:

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

  - 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 $\boldsymbol{A}^\mathsf{T}\boldsymbol{A}$ 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, $\boldsymbol{\beta} = [\beta_0, \beta_1, \beta_2, \beta_3, \beta_4]^\mathsf{T}$ and the rows of $\boldsymbol{A}$ 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 $\boldsymbol{\beta} = [-1, 0, 1, 1, 1]^\mathsf{T}$ yields $\boldsymbol{A}\boldsymbol{\beta} = \mathbf{0}$, as well as $\boldsymbol{A}^\mathsf{T}\boldsymbol{A}\boldsymbol{\beta} = \mathbf{0}$
  - Because there exists $\boldsymbol{\beta} \neq \mathbf{0}$ such that $\boldsymbol{A}^\mathsf{T}\boldsymbol{A}\boldsymbol{\beta} = \mathbf{0}$, we know $\boldsymbol{A}^\mathsf{T}\boldsymbol{A}$ is singular

- To circumvent this problem, . . .
  - Use a non-redundant one-hot coding scheme, e.g.,

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

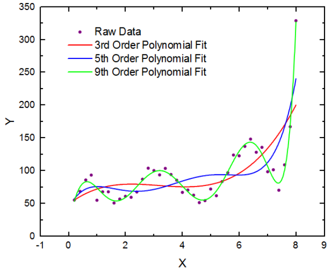
  - Use regularization (see Unit 4)

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 $j = 1, \ldots, d$
- 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