# Unit 2
# Multiple Linear Regression


Prof. Phil Schniter

The Ohio State University


ECE 4300: Introduction to Machine Learning, Sp20

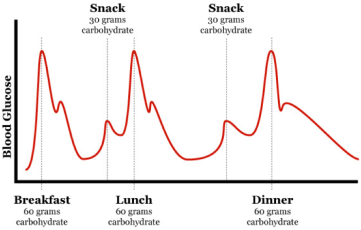
# Learning objectives

- 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 $\mathrm{RSS}$
  - 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

# Outline

# Example: Understanding glucose levels in diabetes patients

- Diabetes patients must monitor their blood glucose level
- What causes glucose levels to rise and fall?
  - Many factors
  - We know some qualitative mechanisms
  - But quantitative models are difficult to obtain
    - Hard to derive from first principles
    - Difficult to model physiological processes
- Can machine learning help?



# Diabetes dataset

- Data was collected as series of events
  - eating
  - exercise
  - insulin dosage
- Glucose level (our target variable) was monitored

About Citation Policy

UCI

Machine Learning Repository

Center for Machine Learning and Intelligent Systems

## Diabetes Data Set

*Download*: Data Folder, Data Set Description

**Abstract**: This diabetes dataset is from AIM '94

| | | | | | |
|---|---|---|---|---|---|
| **Data Set Characteristics:** | Multivariate, Time-Series | **Number of Instances:** | N/A | **Area:** | Life |
| **Attribute Characteristics:** | Categorical, Integer | **Number of Attributes:** | 20 | **Date Donated** | N/A |
| **Associated Tasks:** | N/A | **Missing Values?** | N/A | **Number of Web Hits:** | 161379 |

### Data Set Information:

Diabetes patient records were obtained from two sources: ar
clock to timestamp events, whereas the paper records only p
assigned to breakfast (08:00), lunch (12:00), dinner (18:00),
records have more realistic time stamps.

Diabetes files consist of four fields per record. Each field is s

File Names and format:
(1) Date in MM-DD-YYYY format
(2) Time in XX:YY format
(3) Code
(4) Value

The Code field is deciphered as follows:

33 = Regular insulin dose
34 = NPH insulin dose
35 = UltraLente insulin dose
48 = Unspecified blood glucose measurement
57 = Unspecified blood glucose measurement
58 = Pre-breakfast blood glucose measurement
59 = Post-breakfast blood glucose measurement
60 = Pre-lunch blood glucose measurement
61 = Post-lunch blood glucose measurement
62 = Pre-supper blood glucose measurement
63 = Post-supper blood glucose measurement

# 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

```python
from sklearn import datasets, linear_model, preprocessing

# Load the diabetes dataset
diabetes = datasets.load_diabetes()
X = diabetes.data
y = diabetes.target
```

```python
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 $i$th row is $\boldsymbol{x}_i^\mathsf{T}$, which contains the feature data for the $i$th sample
- The target vector $\boldsymbol{y}$ is in $\mathbb{R}^{n\times 1}$
- The $i$th data sample is the pair $(\boldsymbol{x}_i, y_i)$

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

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

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

# Matrix review

Consider

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} 2 & 0 \\ 3 & 1 \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

- Matrix-vector multiply: $\boldsymbol{A}\boldsymbol{x} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 \cdot 2 + 2 \cdot 3 \\ 3 \cdot 2 + 4 \cdot 3 \\ 5 \cdot 2 + 6 \cdot 3 \end{bmatrix} = \begin{bmatrix} 8 \\ 18 \\ 28 \end{bmatrix}$
- Matrix multiply: $\boldsymbol{A}\boldsymbol{B} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 3 & 1 \end{bmatrix} = \begin{bmatrix} 8 & 1 \cdot 0 + 2 \cdot 1 \\ 18 & 3 \cdot 0 + 4 \cdot 1 \\ 28 & 5 \cdot 0 + 6 \cdot 1 \end{bmatrix} = \begin{bmatrix} 8 & 2 \\ 18 & 4 \\ 28 & 6 \end{bmatrix}$
- Solving a system of linear equations: $\boldsymbol{x} = \boldsymbol{B}\boldsymbol{u}$, $\boldsymbol{u} = \boldsymbol{B}^{-1}\boldsymbol{x}$ if $\boldsymbol{B}$ is invertible
- Matrix inverse: $\boldsymbol{B}^{-1} = \begin{bmatrix} 2 & 0 \\ 3 & 1 \end{bmatrix}^{-1} = \frac{1}{2 \cdot 1 - 3 \times 0} \begin{bmatrix} 1 & -0 \\ -3 & 2 \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ -1.5 & 1 \end{bmatrix}$
- Matrix transpose: $\boldsymbol{A}^{\mathsf{T}} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$. Also, $(\boldsymbol{A}\boldsymbol{B})^{\mathsf{T}} = \boldsymbol{B}^{\mathsf{T}}\boldsymbol{A}^{\mathsf{T}}$.

# 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 $d+1$ terms 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 $i$th sample is

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

- Let's define the 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

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

# Slopes and intercept

- Recall the linear prediction

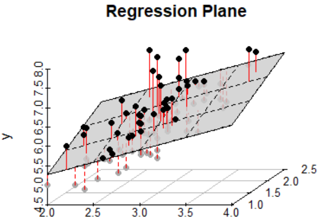
$$\widehat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_d x_d$$

- 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, $\beta_0$ is the intercept
  - $\boldsymbol{\beta}_{1:d}$ contains slope coefficients

- With this notation, we can write

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

which will sometimes be convenient.



# Outline

# The least-squares problem

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

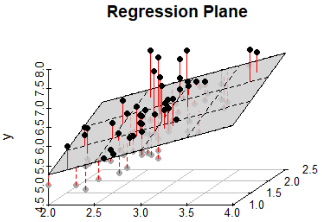
$$\widehat{y}_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_d x_{id}$$

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

- In particular, 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



# The optimization approach: A general ML recipe

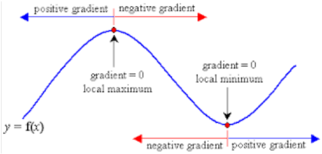
| General ML problem | | Multiple Linear Regression |
|---|---|---|
| Assume a model with some parameters | $\rightarrow$ | Linear model: $\widehat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_d x_d$ |
| Get data | $\rightarrow$ | Data: $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ |
| Choose a loss function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

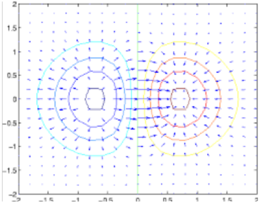
# Gradients of multi-variable functions

- Consider a scalar-valued function $f(\boldsymbol{x}) = f(x_1, \ldots, x_d)$
- $\boxed{\text{If } \boldsymbol{x} \text{ is a local minimum, then } \nabla f(\boldsymbol{x}) = \mathbf{0}}$
- Here, $\nabla f(\boldsymbol{x})$ denotes the gradient of $f$ at $\boldsymbol{x}$:

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} \partial f(\boldsymbol{x})/\partial x_1 \\ \vdots \\ \partial f(\boldsymbol{x})/\partial x_d \end{bmatrix}$$

- The gradient tells the direction and slope of maximum increase
- Ex: If $f(x_1, x_2) = x_1 \sin x_2 + x_1^2 x_2$ then $\nabla f(\boldsymbol{x}) = \left[\begin{smallmatrix} \sin x_2 + 2x_1x_2 \\ x_1 \cos x_2 + x_1^2 \end{smallmatrix}\right]$



# The least-squares solution

- Writing the $\mathrm{RSS}$ 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 $d > n$ then $\boldsymbol{A}^{\mathsf{T}}\boldsymbol{A}$ isn't invertible. We'll talk about this later.

# $R^2$ Goodness-of-fit

- Key question: How good is this linear prediction?

- Let's split the variance-of-$y$ into two parts:

$$s_y^2 = \underbrace{\left[s_y^2 - \frac{\text{RSS}}{n}\right]}_{\text{explained by } \boldsymbol{x}} + \underbrace{\left[\frac{\text{RSS}}{n}\right]}_{\text{unexplained by } \boldsymbol{x}} = \left(\underbrace{\left[1 - \frac{\text{RSS}/n}{s_y^2}\right]}_{\text{explained by } \boldsymbol{x}} + \underbrace{\left[\frac{\text{RSS}/n}{s_y^2}\right]}_{\text{unexplained by } \boldsymbol{x}}\right) s_y^2$$

- The *fraction* of the variance-of-$y$ explained by $\boldsymbol{x}$ is

$$\boxed{1 - \frac{\text{RSS}/n}{s_y^2} \triangleq R^2,}$$ known as the "coefficient of determination"

  - Note that $R^2 \in [0, 1]$
  - Ex: $R^2 = 0.48$ means that "48% of $s_y^2$ is explained by $\boldsymbol{x}$."

- Interpretation: $\begin{cases} R^2 \approx 1: & \text{linear model provides a very good fit} \\ R^2 \approx 0: & \text{linear model provides a very poor fit} \end{cases}$

# RSS as a norm on the vector of residuals

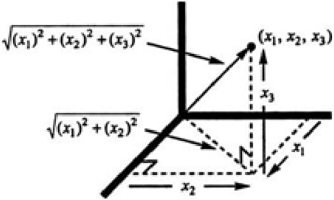
- Recall that the RSS is defined as

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

- 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 $\ell_2$ norm




- This allows us to write the RSS as

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

# The minimum RSS

The minimum $\mathrm{RSS}$ equals

$$\mathrm{RSS}_{\mathsf{min}} = \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{\beta}_{\mathsf{ls}}\|^2 \quad \text{with} \quad \boldsymbol{\beta}_{\mathsf{ls}} = (\boldsymbol{A}^\mathsf{T}\boldsymbol{A})^{-1}\boldsymbol{A}^\mathsf{T}\boldsymbol{y}$$

Recalling that

$$\|\boldsymbol{\epsilon}\|^2 = \sum_i \epsilon_i^2 = \boldsymbol{\epsilon}^\mathsf{T}\boldsymbol{\epsilon} \quad \text{and} \quad (\boldsymbol{B}\boldsymbol{y})^\mathsf{T} = \boldsymbol{y}^\mathsf{T}\boldsymbol{B}^\mathsf{T}$$

we have

$$\begin{aligned}
\mathrm{RSS}_{\mathsf{min}} &= \|\boldsymbol{y} - \boldsymbol{A}(\boldsymbol{A}^\mathsf{T}\boldsymbol{A})^{-1}\boldsymbol{A}^\mathsf{T}\boldsymbol{y}\|^2 \ \text{ assuming } \boldsymbol{A}^\mathsf{T}\boldsymbol{A} \text{ is invertible} \\
&= \|\big(\boldsymbol{I} - \boldsymbol{A}(\boldsymbol{A}^\mathsf{T}\boldsymbol{A})^{-1}\boldsymbol{A}^\mathsf{T}\big)\boldsymbol{y}\|^2 \\
&= \boldsymbol{y}^\mathsf{T}\big(\boldsymbol{I} - \boldsymbol{A}(\boldsymbol{A}^\mathsf{T}\boldsymbol{A})^{-1}\boldsymbol{A}^\mathsf{T}\big)^\mathsf{T}\big(\boldsymbol{I} - \boldsymbol{A}(\boldsymbol{A}^\mathsf{T}\boldsymbol{A})^{-1}\boldsymbol{A}^\mathsf{T}\big)\boldsymbol{y} \\
&= \boldsymbol{y}^\mathsf{T}\big(\boldsymbol{I} - 2\boldsymbol{A}(\boldsymbol{A}^\mathsf{T}\boldsymbol{A})^{-1}\boldsymbol{A}^\mathsf{T} + \boldsymbol{A}(\boldsymbol{A}^\mathsf{T}\boldsymbol{A})^{-1}\boldsymbol{A}^\mathsf{T}\boldsymbol{A}(\boldsymbol{A}^\mathsf{T}\boldsymbol{A})^{-1}\boldsymbol{A}^\mathsf{T}\big)\boldsymbol{y} \\
&= \boldsymbol{y}^\mathsf{T}\big(\boldsymbol{I} - \boldsymbol{A}(\boldsymbol{A}^\mathsf{T}\boldsymbol{A})^{-1}\boldsymbol{A}^\mathsf{T}\big)\boldsymbol{y}
\end{aligned}$$

# Outline

# The LS solution via auto- & cross-correlation

- Recall that the $i$th data sample involves $y_i$ and the $i$th row of $\boldsymbol{A}$:

$$\boldsymbol{a}_i^\mathsf{T} = [a_{i0}, \ldots, a_{id}] = [1, x_{i1}, \ldots, x_{id}]$$

- Let us define the sample auto-correlation matrix

$$\boldsymbol{R}_{aa} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{a}_i \boldsymbol{a}_i^\mathsf{T} = \frac{1}{n}\boldsymbol{A}^\mathsf{T}\boldsymbol{A}$$

  - Note that $[\boldsymbol{R}_{aa}]_{l,m} = \frac{1}{n}\sum_{i=1}^{n} a_{il}a_{im}$ is the sample correlation of features $l$ and $m$

- And let us define the sample cross-correlation vector

$$\boldsymbol{r}_{ay} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{a}_i y_i = \frac{1}{n}\boldsymbol{A}^\mathsf{T}\boldsymbol{y}$$

  - Note that $[\boldsymbol{r}_{ay}]_l = \frac{1}{n}\sum_{i=1}^{n} a_{il}y_i$ is the sample correlation of feature $l$ and target

- Then the least-squares solution can be expressed as

$$\boxed{\boldsymbol{\beta}_{\mathsf{ls}} = \boldsymbol{R}_{aa}^{-1}\boldsymbol{r}_{ay}}$$

# Linear regression on mean-removed data

- Until now we used the intercept term $\beta_0$ to compensate for differences between the means of $y$ and $\boldsymbol{x}$.

- An alternative approach: Predict using mean-removed data and no intercept:
  1) Compute the means, $\overline{y}$ and $\overline{\boldsymbol{x}} = [\overline{x}_1, \dots, \overline{x}_d]^{\mathsf{T}}$
  2) Remove the means, giving $\widetilde{\boldsymbol{y}} = \boldsymbol{y} - \mathbf{1}\overline{y}$ and $\widetilde{\boldsymbol{X}} = \boldsymbol{X} - \mathbf{1}\overline{\boldsymbol{x}}^{\mathsf{T}}$, with $\mathbf{1} = [1, \dots, 1]^{\mathsf{T}}$
  3) Predict $\widetilde{\boldsymbol{y}}$ from $\widetilde{\boldsymbol{X}}$ using linear regression without an intercept:
  $$\widetilde{\boldsymbol{y}} \approx \widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{\beta}} \triangleq \widehat{\widetilde{\boldsymbol{y}}} \quad \Rightarrow \quad \widetilde{\boldsymbol{\beta}}_{\mathsf{ls}} = (\widetilde{\boldsymbol{X}}^{\mathsf{T}}\widetilde{\boldsymbol{X}})^{-1}\widetilde{\boldsymbol{X}}^{\mathsf{T}}\widetilde{\boldsymbol{y}}$$
  4) Restore mean when predicting the target $y$ from a new sample $\boldsymbol{x}$. In particular, remove the mean to get $\widetilde{\boldsymbol{x}} \triangleq \boldsymbol{x} - \overline{\boldsymbol{x}}$, then
  $$\begin{aligned} \widehat{y} &= \widehat{\widetilde{y}} + \overline{y} = \widetilde{\boldsymbol{x}}^{\mathsf{T}}\widetilde{\boldsymbol{\beta}}_{\mathsf{ls}} + \overline{y} = \left(\boldsymbol{x}^{\mathsf{T}} - \overline{\boldsymbol{x}}^{\mathsf{T}}\right)\widetilde{\boldsymbol{\beta}}_{\mathsf{ls}} + \overline{y} = \boldsymbol{x}^{\mathsf{T}}\widetilde{\boldsymbol{\beta}}_{\mathsf{ls}} + \left(\overline{y} - \overline{\boldsymbol{x}}^{\mathsf{T}}\widetilde{\boldsymbol{\beta}}_{\mathsf{ls}}\right) \\ &= [1 \ \ \boldsymbol{x}^{\mathsf{T}}] \begin{bmatrix} \beta_0 \\ \boldsymbol{\beta}_{1:d} \end{bmatrix} \text{ with } \beta_0 = \overline{y} - \overline{\boldsymbol{x}}^{\mathsf{T}}\widetilde{\boldsymbol{\beta}}_{\mathsf{ls}} \text{ and } \boldsymbol{\beta}_{1:d} = \widetilde{\boldsymbol{\beta}}_{\mathsf{ls}} \end{aligned}$$

- Can show that $[\beta_0 \ \ \boldsymbol{\beta}_{1:d}^{\mathsf{T}}] = \boldsymbol{\beta}_{\mathsf{ls}}^{\mathsf{T}}$, i.e., the two approaches are equivalent.

# The LS solution via auto- & cross-covariance

- Define the sample auto-covariance matrix and sample cross-covariance vector

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

$$\boldsymbol{s}_{xy} \triangleq \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i - \overline{\boldsymbol{x}})(y_i - \overline{y}) = \tfrac{1}{n}\widetilde{\boldsymbol{X}}^{\mathsf{T}}\widetilde{\boldsymbol{y}}$$

- We know from the previous page that $\widetilde{\boldsymbol{\beta}}_{\mathsf{ls}} = \boldsymbol{S}_{xx}^{-1}\boldsymbol{s}_{xy}$

- Thus we can write the LS prediction coefficients as

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

# Outline

# 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]
```

  Then we design $\boldsymbol{\beta}$ using the training data, and evaluate performance (e.g., $\mathrm{RSS}$) on the test data.
- We will discuss train/test splits in much more detail in the next unit

# Manually computing the LS solution with Numpy

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

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

```
ones = np.ones((ns_train,1))
A = np.hstack((ones,X_tr))
```

but explicitly computing the matrix inverse can be slow

- It is better to attack the LS problem "$\arg\min_{\boldsymbol{\beta}} \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{\beta}\|^2$" directly via

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

where np.linalg.lstsq uses more efficient LAPACK routines.

# Linear regression via sklearn

- A much easier 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. This gives $R^2 = 0.51$.

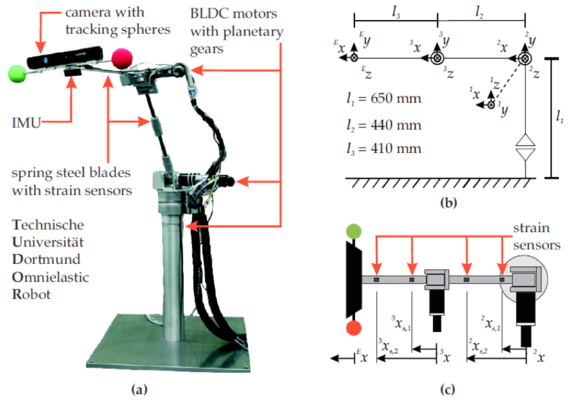
```
regr = linear_model.LinearRegression()
regr.fit(X_tr,y_tr)
```

```
X_test = X[ns_train:,:]
y_test = y[ns_train:]
y_test_pred = regr.predict(X_test)
RSS_test = np.mean((y_test_pred-y_test)**2)
Rsq_test = 1-RSS_test/(np.std(y_test)**2)
print("R^2 =              {0:f}".format(Rsq_test))
```

```
R^2 =              0.507199
```

# Lab: Robot calibration

- Goal: predict current draw (affects power consumption)
- Predictors:
  - Joint angles, velocities, accelerations
  - Strain gauge readings (measure of load)
- More details at http://www.rst.e-technik.tu-dortmund.de/cms/en/research/robotics/TUDOR_engl/index.html



# 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 $d = 1$
- Why use multiple linear regression?

# Special case: Multiple linear regression with $d = 1$

- When $d = 1$ we have $\boldsymbol{A} = \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}$ and $\boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$

- Thus, the LS solution is

$$\begin{aligned}
\boldsymbol{\beta}_{\mathsf{ls}} &= (\tfrac{1}{n}\boldsymbol{A}^{\mathsf{T}}\boldsymbol{A})^{-1}(\tfrac{1}{n}\boldsymbol{A}^{\mathsf{T}}\boldsymbol{y}) = \begin{bmatrix} 1 & \overline{x} \\ \overline{x} & \boldsymbol{x}^{\mathsf{T}}\boldsymbol{x}/n \end{bmatrix}^{-1} \begin{bmatrix} \overline{y} \\ \boldsymbol{x}^{\mathsf{T}}\boldsymbol{y}/n \end{bmatrix} \\
&= \begin{bmatrix} 1 & \overline{x} \\ \overline{x} & s_{xx} + \overline{x}^2 \end{bmatrix}^{-1} \begin{bmatrix} \overline{y} \\ s_{xy} + \overline{xy} \end{bmatrix} = \frac{1}{s_{xx} + \overline{x}^2 - \overline{x}^2} \begin{bmatrix} s_{xx} + \overline{x}^2 & -\overline{x} \\ -\overline{x} & 1 \end{bmatrix} \begin{bmatrix} \overline{y} \\ s_{xy} + \overline{xy} \end{bmatrix} \\
&= \frac{1}{s_{xx}} \begin{bmatrix} s_{xx}\overline{y} + \overline{x}^2\overline{y} - \overline{x}s_{xy} - \overline{x}^2\overline{y} \\ -\overline{xy} + s_{xy} + \overline{xy} \end{bmatrix} = \frac{1}{s_{xx}} \begin{bmatrix} s_{xx}\overline{y} - \overline{x}s_{xy} \\ s_{xy} \end{bmatrix} = \begin{bmatrix} \overline{y} - \overline{x}s_{xy}/s_{xx} \\ s_{xy}/s_{xx} \end{bmatrix}
\end{aligned}$$

- This matches the LS solution $\beta_1 = s_{xy}/s_{xx}$ and $\beta_0 = \overline{y} - \overline{x}\beta_1$ that we derived earlier in the context of simple linear regression

- It's a special case of the co-variance-matrix expression for $\boldsymbol{\beta}_{\mathsf{ls}}$ under general $d$

# 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

```
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
```

# 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,BMW,GM}.
  - Problem: Coding brands as ordinal numbers like $\{1, 2, 3\}$ works poorly. 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 |
| BMW | 0 | 1 | 0 |
| GM | 0 | 0 | 1 |

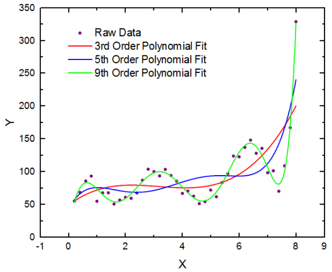
  - Linear model becomes $y \approx \beta_0 + \beta_1 x_1 + \beta_2 x_2^{(1)} + \beta_3 x_2^{(2)} + \beta_4 x_2^{(3)}$
  - Essentially, this gives $3$ *different* linear models:
    - Ford: $y \approx \beta_0 + \beta_1 x_1 + \beta_2$
    - BMW: $y \approx \beta_0 + \beta_1 x_1 + \beta_3$
    - GM: $y \approx \beta_0 + \beta_1 x_1 + \beta_4$
  - Interpretation: One-hot coding implies a different intercept for each category!

# 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



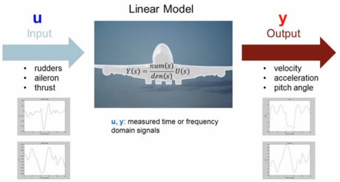

- Easy to handle using multiple linear regression: Simply assign $x_j = x^j$ for $j = 1, \ldots, d$
- Note: same idea can be used for other nonlinear models, not only polynomial!
- Like one-hot coding, this is an instance of feature transformation
- Problem: how do we choose the polynomial order $d$?
  - Will discuss this in the next unit

# Application: Learning a linear time-invariant system

- Auto-regressive moving-average (ARMA) model of an LTI system:

$$y_t \approx a_1 y_{t-1} + \cdots + a_m y_{t-m} + b_0 x_t + b_1 x_{t-1} + \cdots b_n x_{t-n}$$

- Transfer function: $H(z) = \frac{b_0 + b_1 z^{-1} + \cdots b_n z^{-n}}{1 - a_1 z^{-1} - \cdots - a_m z^{-m}}$
- Goal: given inputs $\{x_i\}_{t=1}^T$ and outputs $\{y_t\}_{t=1}^T$, estimate the ARMA parameters $\boldsymbol{\beta} = [a_1, \ldots, a_m, b_0, \ldots, b_n]^\mathsf{T}$
- An instance of multiple linear regression!
  - Write as $\boldsymbol{y} \approx \boldsymbol{A}\boldsymbol{\beta}$ with appropriate definitions of $\boldsymbol{A}$ and $\boldsymbol{y}$
  - See homework problem
- Many engineering applications!
  - learning dynamics of mechanical systems
  - modeling responses in neural systems
  - speech coding: fit a model every 25 ms
  - predicting stock-market time series



# 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 $\mathrm{RSS}$
  - 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