CS-UY 4563: Lecture 3
Multiple Linear Regression

NYU Tandon School of Engineering, Prof. Christopher Musco

COURSE ADMIN

- First lab assignment `lab_housing_partial.ipynb` due **tomorrow, by midnight.**
- First written assignment due **Thursday, by midnight.**
    - 10% extra credit if you use LaTeX (Overleaf is easy) or Markdown (I use Typora) to typeset your assignment.

Training Dataset:

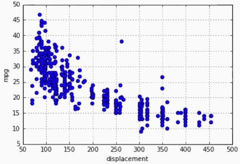
- Given input pairs $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$.
- Each $\mathbf{x}_i$ is an input data point (the predictor).
- Each y_i is a continuous output variable (the target).

Objective:

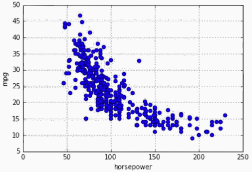
- Have the computer automatically find some function $f(\mathbf{x})$ such that $f(\mathbf{x}_i)$ is close to y_i for the input data.

Predict miles per gallon of a vehicle given information about its engine/make/age/etc.

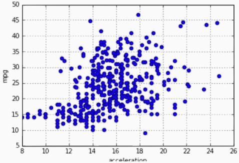
mpg



Displacement



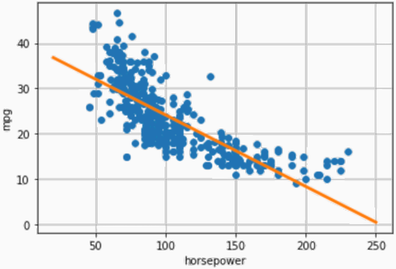
Horsepower



Acceleration

Dataset:

- $x_1, \ldots, x_n \in \mathbb{R}$ (horsepowers of n cars – this is the predictor/independent variable)
- $y_1, \ldots, y_n \in \mathbb{R}$ (MPG – this is the response/dependent variable)



What are the three components needed to setup a supervised learning problem?

1.

2.

3.

- **Model $f_{\boldsymbol{\theta}}(x)$:** Class of equations or programs which map input x to predicted output. We want $f_{\boldsymbol{\theta}}(x_i) \approx y_i$ for training inputs.
- **Model Parameters $\boldsymbol{\theta}$:** Vector of numbers. These are numerical nobs which parameterize our class of models.
- **Loss Function $L(\boldsymbol{\theta})$:** Measure of how well a model fits our data. Typically some function of $f_{\boldsymbol{\theta}}(x_1) - y_1, \ldots, f_{\boldsymbol{\theta}}(x_n) - y_n$

Goal: Choose parameters $\boldsymbol{\theta}^*$ which minimize the Loss Function:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$$

Linear Regression

- Model: $f_{\beta_0,\beta_1}(x) = \beta_0 + \beta_1 \cdot x$
- Model Parameters: β_0, β_1
- Loss Function: $L(\beta_0, \beta_1) = \sum_{i=1}^{n} |y_i - f_{\beta_0,\beta_1}(x_i)|^2$

Goal: Choose β_0, β_1 to minimize
$L(\beta_0, \beta_1) = \sum_{i=1}^{n} |y_i - \beta_0 - \beta_1 x_i|^2$.

Claim: $L(\beta_0, \beta_1)$ is minimized when:

- $\beta_1 = \sigma_{xy}/\sigma_x^2$
- $\beta_0 = \bar{y} - \beta_1\bar{x}$

Where:

- Let $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$. $\bar{y}$ is the <u>mean</u> of y.
- Let $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$. $\bar{y}$ is the <u>mean</u> of x.
- Let $\sigma_x^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$. σ_x^2 is the <u>variance</u> of x.
- Let $\sigma_{xy} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$. σ_{xy} is the <u>covariance</u>.

Note: Only got a nice closed form solution thanks to our choice of loss function.

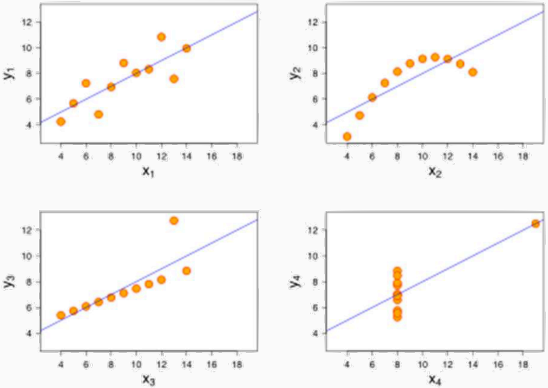
Let $L_{\text{min}} = \min_{\beta_0,\beta_1} L(\beta_0, \beta_1)$.

$$R^2 = 1 - \frac{L_{\text{min}}}{n\sigma_y^2}$$

is exactly the R^2 value you may remember from statistics. A.k.a. the **"coefficient of determination"**.

The smaller the loss, the closer R^2 is to 1, which means we have a better regression fit.

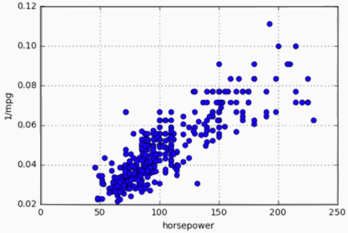
Many reasons you might get a poor regression fit:



Some of these are fixable!

- Remove outliers, use more robust loss function.
- **Non-linear model transformation.**

Fit the model $\frac{1}{\text{mpg}} \approx \beta_0 + \beta_1 \cdot \text{horsepower}$.

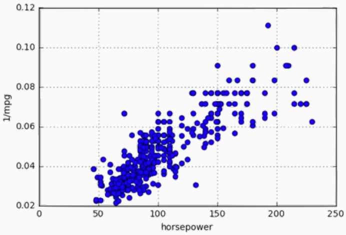


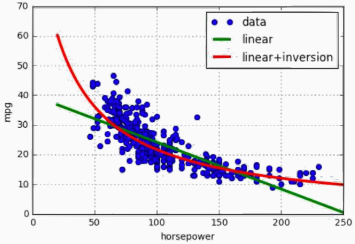
Fit the model $\frac{1}{\text{mpg}} \approx \beta_0 + \beta_1 \cdot \text{horsepower}$.

- Set $\tilde{y}_1, \ldots, \tilde{y}_n = 1/y_1, \ldots, 1/y_n$.
- Learn function f such that $f(\mathbf{x}_i)$ predicts $\tilde{y}_i$.
- Predict $1/f(\mathbf{x}_i)$ as MPG for car i.

Fit the model $\frac{1}{\text{mpg}} \approx \beta_0 + \beta_1 \cdot$ horsepower.

- Set $\tilde{y}_1, \ldots, \tilde{y}_n = 1/y_1, \ldots, 1/y_n$.
- Learn function f such that $f(\mathbf{x}_i)$ predicts $\tilde{y}_i$.
- Predict $1/f(\mathbf{x}_i)$ as MPG for car i.





Much better fit, same exact learning algorithm!

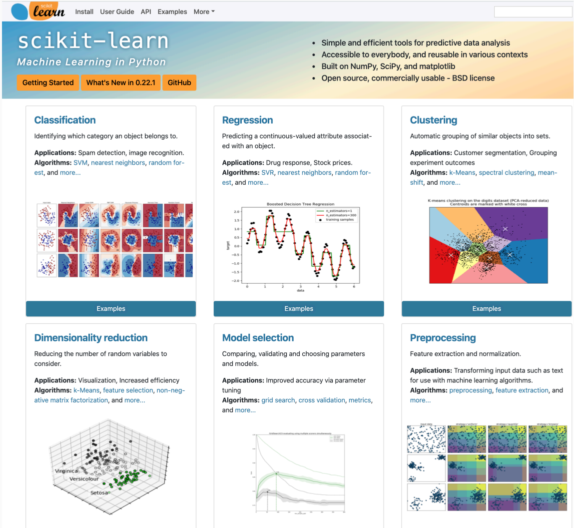
Predict target y using multiple features, simultaneously.

Motivating example: Predict diabetes progression in patients after 1 year based on health metrics. (Measured via numerical score.)

Features: Age, sex, body mass index, average blood pressure, six blood serum measurements (e.g. cholesterol, lipid levels, iron, etc.)

Demo in `demo1_diabetes.ipynb`.

Introducing **Scikit Learn**.



Pros:

- One of the most popular “traditional” ML libraries.
- Many built in models for regression, classification, dimensionality reduction, etc.
- Easy to use, works with ‘numpy‘, ‘scipy‘, other libraries we use.
- Great for rapid prototyping, testing models.

Cons:

- Everything is very “black-box”: difficult to debug, understand why models aren’t working, speed up code, etc.
- You will likely want to dive deeper than the built-in functions for your project.

Modules used:

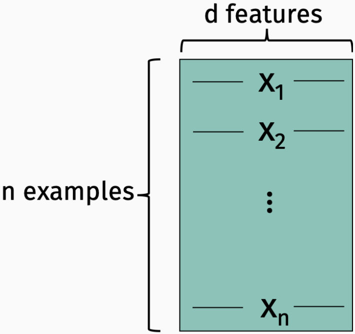
- `datasets` module contains a number of pre-loaded datasets. Saves time over downloading and importing with `pandas`.
- `linear_model` can be used to solve Multiple Linear Regression. A bit overkill for this simple model, but gives you an idea of `sklearn`'s general structure.

Target variable:

- Scalars $y_1, \ldots, y_n$ for n data examples (a.k.a. samples).

Predictor variables:

- d dimensional vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ for n data examples and d features

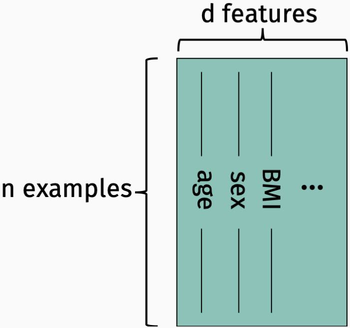


Target variable:

- Scalars $y_1, \ldots, y_n$ for n data examples (a.k.a. samples).

Predictor variables:

- d dimensional vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ for n data examples and d features



Data matrix indexing:

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1d} \\ x_{21} & x_{22} & \dots & x_{2d} \\ x_{31} & x_{32} & \dots & x_{3d} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nd} \end{bmatrix}$$

Multiple Linear Regression Model:

$$\text{Predict} \qquad y_i \approx \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_d x_{id}$$

The rate at which diabetes progress depends on many factors, with each factor having a different magnitude effect.

Assume first columns contains all 1's. If it doesn't append on a column of all 1's.

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1d} \\ x_{21} & x_{22} & \dots & x_{2d} \\ x_{31} & x_{32} & \dots & x_{3d} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nd} \end{bmatrix} = \begin{bmatrix} 1 & x_{12} & \dots & x_{1d} \\ 1 & x_{22} & \dots & x_{2d} \\ 1 & x_{32} & \dots & x_{3d} \\ \vdots & \vdots & & \vdots \\ 1 & x_{n2} & \dots & x_{nd} \end{bmatrix}$$

Multiple Linear Regression Model:

Predict $\quad y_i \approx \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_d x_{id}$

Use as much linear algebra notation as possible!

- Model:
- Model Parameters:
- Loss Function:

Linear Least-Squares Regression.

- Model:

$$f_{\boldsymbol{\beta}}(\mathbf{x}) = \langle \mathbf{x}, \boldsymbol{\beta} \rangle$$

- Model Parameters:

$$\boldsymbol{\beta} = [\beta_1, \beta_2, \ldots, \beta_d]$$

- Loss Function:

$$\begin{aligned} L(\boldsymbol{\beta}) &= \sum_{i=1}^{n} |y_i - \langle \mathbf{x}_i, \boldsymbol{\beta} \rangle|^2 \\ &= \|\mathbf{y} - \mathbf{X}\beta\|_2^2 \end{aligned}$$

Machine learning goal: minimize the loss function $L(\boldsymbol{\beta}) : \mathbb{R}^d \to \mathbb{R}$.

Find optimum by determining for which $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_d]$ all partial derivatives are 0. I.e. when do we have:

$$\begin{bmatrix} \frac{\partial L}{\partial \beta_1} \\ \frac{\partial L}{\partial \beta_2} \\ \vdots \\ \frac{\partial L}{\partial \beta_d} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \ldots \\ 0 \end{bmatrix}$$

For any function $L(\boldsymbol{\beta}) : \mathbb{R}^d \to \mathbb{R}$, $\nabla L(\beta)$ is a function from $\mathbb{R}^d \to \mathbb{R}^d$ defined:

$$\nabla L(\beta) = \begin{bmatrix} \frac{\partial L}{\partial \beta_1} \\ \frac{\partial L}{\partial \beta_2} \\ \vdots \\ \frac{\partial L}{\partial \beta_d} \end{bmatrix}$$

The <u>gradient</u> of the loss function is a central tool in machine learning. We will use it again and again.

Loss function:

$$\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2$$

Gradient:

$$-2 \cdot \mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$$

GRADIENT WARMUP

Loss function: $\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2$.

Goal: minimize the loss function $L(\boldsymbol{\beta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2$.

$$-2 \cdot \mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) = \mathbf{0}$$

Solve for optimal $\boldsymbol{\beta}^*$:

$$\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}^* = \mathbf{X}^T\mathbf{y}$$

$$\boldsymbol{\beta}^* = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

Multiple linear regression solution

Need to compute $\beta^* = \arg\min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$.

- Main cost is computing $(\mathbf{X}^T\mathbf{X})^{-1}$ which takes $O(nd^2)$ time.
- Can solve slightly faster using the method `numpy.linalg.lstsq`, which is running an algorithm based on QR decomposition.
- For larger problems, can solve much faster using an *iterative methods* like `scipy.sparse.linalg.lsqr`.

Will learn more about iterative methods when we study Gradient Descent.

What is the sign of β_1 when we run a simple linear regression using the following predictors in isolation:

- Body mass index (BMI): **positive**
- Sex (values of 1 indicates male, value of 2 indicates female): **positive**

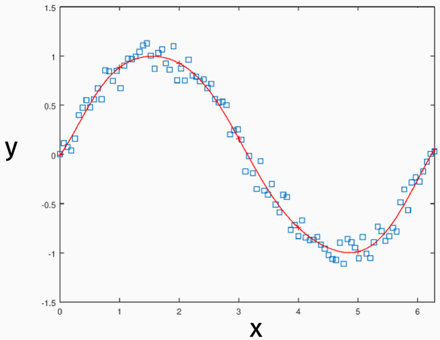
What is the sign of the corresponding β's when we run a multiple linear regression using the following predictors together:

- Body mass index (BMI): **positive**
- Sex (values of 1 indicates male, value of 2 indicates female): **negative**

Can you explain this? What are other examples when this phenomenon might show up?

How could we fit the non-linear model:

$$y_i \approx \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3.$$



Transform into a multiple linear regression problem:

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^1 & x_2^3 \\ 1 & x_3 & x_3^2 & x_3^3 \\ \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & x_n^3 \end{bmatrix}$$

Each column j is generated by a different basis function $\phi_j(x)$.

Could have:

- $\phi_j(x) = x^q$
- $\phi_j(x) = sin(x)$
- $\phi_j(x) = cos(10)$
- $\phi_j(x) = 1/x$

Suppose we go back to the MPG prediction problem. What if we had a <u>categorical</u> random variable for car make: e.g. Ford, BMW, Honda. **How would you encode as a numerical column?**

$$\begin{bmatrix} \texttt{ford} \\ \texttt{ford} \\ \texttt{honda} \\ \texttt{bmw} \\ \texttt{honda} \\ \texttt{ford} \end{bmatrix} \rightarrow \begin{bmatrix} \quad \\ \\ \\ \\ \\ \quad \end{bmatrix}$$

Better approach: <u>One Hot Encoding.</u>

$$\begin{bmatrix} \texttt{ford} \\ \texttt{ford} \\ \texttt{honda} \\ \texttt{bmw} \\ \texttt{honda} \\ \texttt{ford} \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Avoids adding inadvertent linear relationships.