

Regression and Gradient Descent

Prof. Alessandro Lucantonio

Aarhus University

24/10/2023

Weight-Height dataset

Task: build a model that predicts the height given the weight.

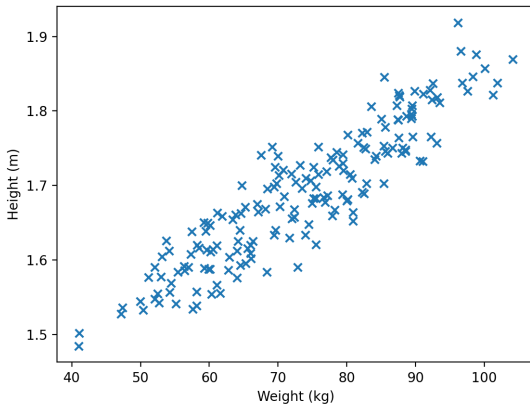


Figure: Plot of the dataset

A solution - Linear Regression model

Remarks:

- ▶ Regression problem (continuous output).
- ▶ Data with different orders of magnitude.

A possible solution to this problem is a linear model (red line in the figure below). This learning algorithm is called **Linear Regression** (LR).

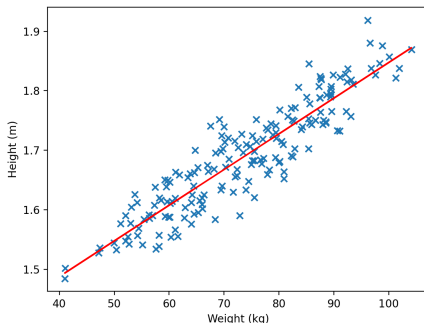


Figure: Linear regression model.

Linear Regression: main ingredients

Notation:

- ▶ $x^{(i)}$: a data sample (weight of the i -th person).
- ▶ $y^{(i)}$: the target corresponding to $x^{(i)}$ (height).
- ▶ N : number of samples.

Model/hypothesis:

$$h_{\mathbf{w}}(x^{(i)}) = w_1 x^{(i)} + w_0$$

where $\mathbf{w} = [w_0, w_1]^T$ is the vector of parameters to be learned.

The set $\mathcal{H} := \{h_{\mathbf{w}} | \mathbf{w} \in \mathbb{R}^2\}$ is called **hypothesis space**.

How to learn \mathbf{w} from data?

Performance measure: Mean Squared Error (MSE)

To evaluate how good is the prediction we compute the **Mean Squared Error** (MSE) is:

$$E(\mathbf{w}) := \frac{1}{N} \sum_{i=1}^N (h_{\mathbf{w}}(x^{(i)}) - y^{(i)})^2.$$

To find the “best” set of parameters, we minimize the MSE:

$$\mathbf{w} \in \arg \min_{\tilde{\mathbf{w}} \in \mathbb{R}^2} E(\tilde{\mathbf{w}}).$$

Multiple-feature Linear Regression

- ▶ Dataset: $\mathbf{x}^{(i)} \in \mathbb{R}^n, y^{(i)} \in \mathbb{R}$, where $x_j^{(i)}$ is the j -th feature of the i -th sample and n is the number of features.
- ▶ Hypothesis:

$$\begin{aligned}h_{\mathbf{w}}(\mathbf{x}^{(i)}) &= w_n x_n^{(i)} + w_{n-1} x_{n-1}^{(i)} + \cdots + w_1 x_1^{(i)} + w_0 \\&= \sum_{i=0}^n w_i \tilde{x}_i^{(i)} = \mathbf{w}^T \tilde{\mathbf{x}}^{(i)},\end{aligned}$$

where $\mathbf{w} = [w_0, \dots, w_n]^T$ and $\tilde{\mathbf{x}}^{(i)} = [1, x_1^{(i)}, \dots, x_n^{(i)}]^T$.

Multiple-feature Linear Regression - MSE

MSE:

$$\begin{aligned} E(\mathbf{w}) &= \frac{1}{N} \sum_{i=1}^N (h_{\mathbf{w}}(\mathbf{x}^{(i)}) - y^{(i)})^2 \\ &= \frac{1}{N} (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y}) \\ &= \frac{1}{N} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 \end{aligned}$$

where

$$\mathbf{X} = \begin{bmatrix} (\tilde{\mathbf{x}}^{(1)})^T \\ \vdots \\ (\tilde{\mathbf{x}}^{(N)})^T \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{bmatrix}.$$

Notice that $\mathbf{X} \in \mathbb{R}^{N \times (n+1)}$ and $\mathbf{y} \in \mathbb{R}^N$.

Coefficient of determination

Idea: X and \mathbf{y} can be thought as two random variables.

Let $\epsilon := \mathbf{y} - X\mathbf{w}$ the vector of the residuals and $\sigma_{\mathbf{y}} := \sqrt{\text{Var}(\mathbf{y})}$ the standard deviation of the targets. The quantity

$$R^2 := 1 - \frac{\|\epsilon\|^2}{\sigma_{\mathbf{y}}^2}$$

is called the *coefficient of determination*.

Best case scenario: $\sigma_{\epsilon} = 0$, hence $R^2 = 1$.

Finding the minimum: Gradient Descent

How to find $\mathbf{w} \in \arg \min_{\tilde{\mathbf{w}} \in \mathbb{R}^2} E(\tilde{\mathbf{w}})$?

Main idea: geometrically, the gradient of a scalar function represents the direction of maximum slope. Hence, following the direction opposite to the gradient allows to decrease the value of the function, *i.e.*

$$E(\mathbf{w}^{j+1}) \leq E(\mathbf{w}^j)$$

Formally:

- ▶ Start with an initial guess \mathbf{w}^0 .
- ▶ For $j \geq 0$, update $\mathbf{w}^{j+1} := \mathbf{w}^j + \mathbf{d}^j$, where \mathbf{d}^j is such that

$$\mathbf{d}^j = -\alpha \nabla E(\mathbf{w}^j)$$

where $\alpha > 0$ is the **learning rate**.

Gradient Descent - 3D visualization

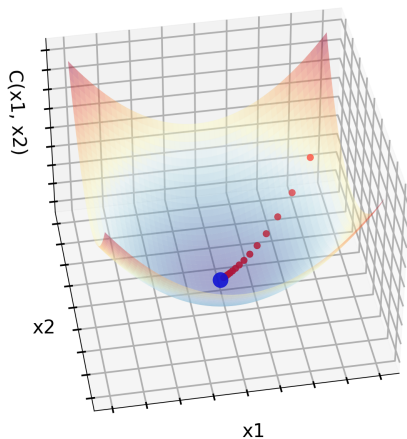


Figure: n blue the global minimum, in red the iteration points.

Gradient Descent: Effect of the learning rate/1

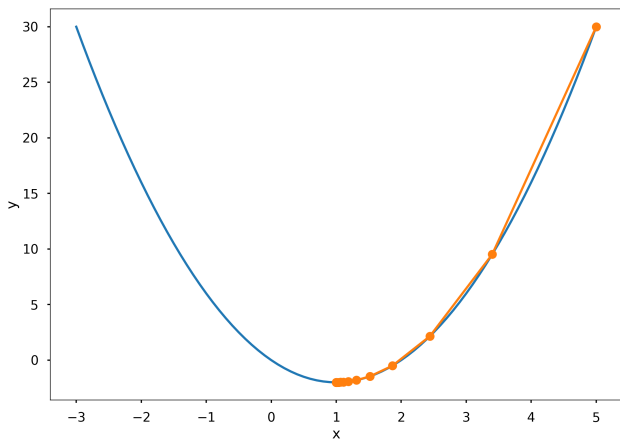


Figure: Learning rate = 0.1

Gradient Descent: Effect of the learning rate/2

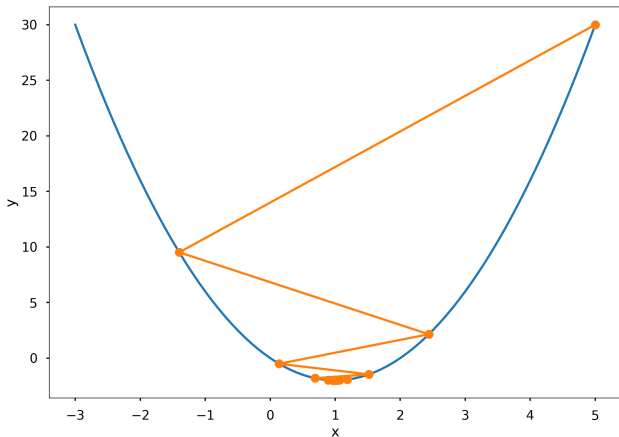


Figure: Learning rate = 0.4

Gradient Descent: Effect of the learning rate/3

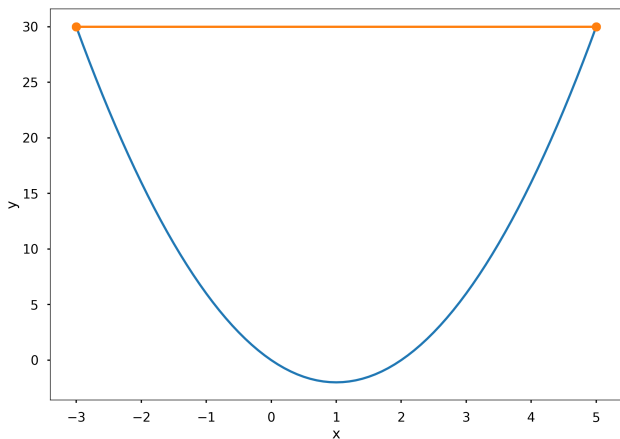


Figure: Learning rate = 0.5

Batch GD, SGD and Mini-Batch GD - Intuition

Notation: $E(\mathbf{w}) = 1/N \sum_{i=1}^N E_i(\mathbf{w})$

The classical gradient descent update rule, *i.e.* update the weights computing the gradient of the entire cost (error) function $E(\mathbf{w})$, is called **batch version**. However, for large number of samples (N) computing $\nabla E(\mathbf{w})$ is very time consuming.

To speed-up the update rule we approximate $\nabla E(\mathbf{w})$ with $\nabla E_i(\mathbf{w})$. This is the idea behind the so-called **Stochastic Gradient Descent** (SGD) or **online version**.

A trade-off between batch GD and SGD is called the **mini-batch** GD.

Batch GD, SGD and Mini-Batch GD - Algorithms

Batch GD

- ▶ Start with an initial guess \mathbf{w}^0 .
- ▶ For $j \geq 0$, update $\mathbf{w}^{j+1} := \mathbf{w}^j - \alpha \nabla E(\mathbf{w}^j)$.

SGD (online)

- ▶ Start with an initial guess \mathbf{w}^0 .
- ▶ For each epoch $j \geq 0$:
 - ▶ draw a random sample i from the dataset;
 - ▶ for each $1 \leq i \leq N$ update $\mathbf{w} = \mathbf{w} - \alpha \nabla E_i(\mathbf{w})$.

Mini-Batch GD

- ▶ Fix an integer $1 \leq \text{mb} \leq N$ (mini-batch size).
- ▶ Start with an initial guess \mathbf{w}^0 .
- ▶ For each epoch $j \geq 0$:
 - ▶ draw a random batch from the dataset;
 - ▶ for each $0 \leq i < \frac{N}{\text{mb}}$ update

$$\mathbf{w} := \mathbf{w} - \alpha \nabla \sum_{k=i \cdot \text{mb} + 1}^{(i+1) \cdot \text{mb}} E_k(\mathbf{w}).$$

Tips and Tricks - How to choose?

- ▶ Batch: usually more stable and provide a more accurate estimation of the gradient, but slow.
- ▶ SGD: fast, stochastic approximation of the gradient implies possible instability (zig-zag effect)
- ▶ Mini-Batch GD: a trade-off between Batch GD and SGD (parallelism available).

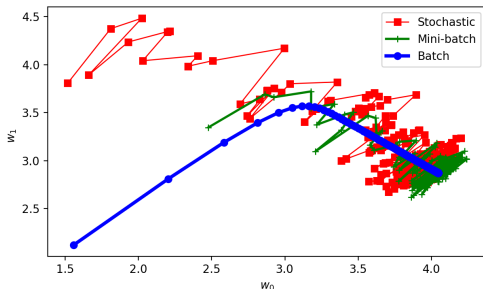


Figure: Batch GD vs SGD vs Mini-Batch GD

Normal Equation for LR and Gradient Descent

We have $E(\mathbf{w}) = \frac{1}{N} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$, hence

$$\nabla E(\mathbf{w}) = \frac{1}{N} \nabla (\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2) = \frac{2}{N} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y})$$

- Normal equation (\iff holds if $\mathbf{X}^T \mathbf{X}$ is invertible):

$$\begin{aligned} \nabla E(\mathbf{w}) = 0 &\iff \frac{2}{N} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) = 0 \\ &\iff \mathbf{X}^T \mathbf{X} \mathbf{w} = \mathbf{X}^T \mathbf{y} \\ &\iff \mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \end{aligned}$$

- Gradient descent main iteration for LR:

$$\mathbf{w}^{j+1} := \mathbf{w}^j - \frac{2\alpha}{N} \mathbf{X}^T (\mathbf{X}\mathbf{w}^j - \mathbf{y})$$

Tips and Tricks - Invertibility of $X^T X$

Invertibility of $X^T X \iff$ columns of X linearly independent.

What if $X^T X$ is not invertible?

If two columns are linearly dependent, then those features are correlated (**redundant**).

Solution: discard one of those features.

Normal Equation vs Gradient Descent

Normal equation:

- ▶ No hyperparameters (explicit solution).
- ▶ No iterations.
- ▶ $\mathcal{O}(N^3)$, since this is the cost to invert a dense matrix. In particular, it is slow when N is large.

Gradient Descent:

- ▶ Need to choose the learning rate α .
- ▶ Needs many iterations.
- ▶ $\mathcal{O}(N^2)$, hence faster when N is large.

Tips and Tricks - Standardization

General (not only for LR): features must have similar magnitudes!

- ▶ Speed up the convergence of gradient descent.
- ▶ Try to have (on average) $-1 \leq \mathbf{x}^{(i)} \leq 1$.

Common techniques:

- ▶ **Feature scaling.** Compute the feature max $\mathbf{M} := [\max_i x_j^{(i)}]$ and the feature min $\mathbf{m} := [\min_i x_j^{(i)}]$ vectors. Then normalize features as follows

$$\mathbf{x}_{\text{norm}}^i = \frac{\mathbf{x}^{(i)} - \mathbf{m}}{\mathbf{M} - \mathbf{m}}$$

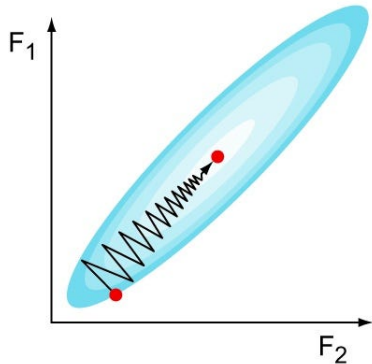
- ▶ **Mean normalization.** Compute the feature mean μ ($\mu_j := \mathbb{E}[[x_j^{(i)}]_i]$) and the feature standard deviation σ ($\sigma_j := \sqrt{\text{Var}[[x_j^{(i)}]_i]}$). Then normalize features as follows

$$\mathbf{x}_{\text{norm}}^{(i)} = \frac{\mathbf{x}^{(i)} - \mu}{\sigma}$$

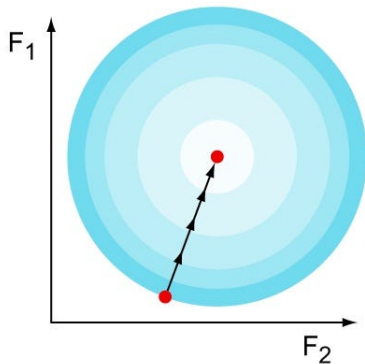
Tips and Tricks - Standardization

Gradient descent with and without feature scaling

Non-normalized features



Normalized features



Polynomial regression (PR)

PR corresponds to polynomial hypothesis, i.e. of the form

$$h_{\mathbf{w}}(\mathbf{x}) = \sum_{j=0}^n w_j x_j^j.$$

More in general: linear basis expansion (LBE)

$$h_{\mathbf{w}}(\mathbf{x}) = \sum_{j=0}^n w_j \phi_j(\mathbf{x}),$$

where $\phi_j : \mathbb{R}^n \rightarrow \mathbb{R}$.