Linear Regression

Prof. Alessandro Lucantonio

Aarhus University - Department of Mechanical and Production Engineering

?/?/2023

Weight-Height example

Dataset: heights and weights of different people.

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

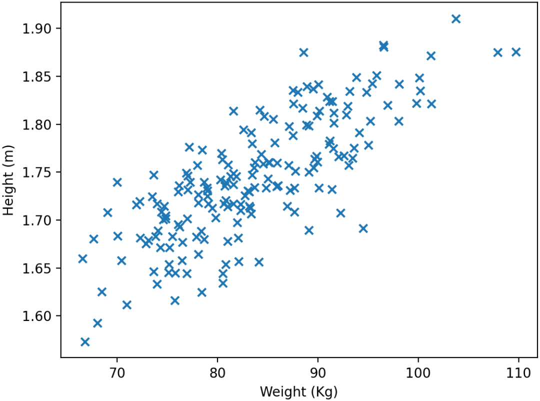


Figure: Data plot

A solution - Linear regression model

Some remarks on data.

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

A possible solution to this problem is represented by **linear regression** (LR).

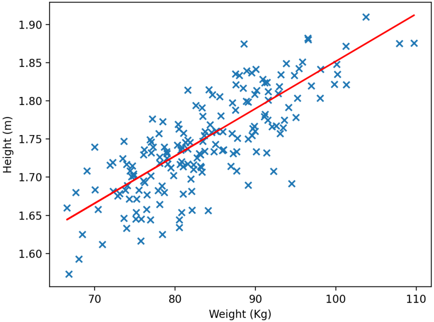


Figure: Trained model (in red)

General ingredients

Notation:

- x: a data sample.
- y: the data target corresponding to x
- N: number of data.

Model/hypothesis: $h_{\boldsymbol{w}}(x) = w_1 x + w_0$, where $\boldsymbol{w} = [w_0, w_1]$ is the vector of parameter that has to be learned.
In our example, x is the weight of a single sample and $h_{\boldsymbol{w}}(x)$ corresponds to the prediction of its height.

Usually the vector $\boldsymbol{w}$ is called **weights vector** and the set $\mathcal{H} := \{h_{\boldsymbol{w}} | \boldsymbol{w} \in \mathbb{R}^2\}$ is called **hypothesis space**.

How to learn w from data?

Mean squared error (MSE)

Given a training sample x_i and a model $h_{\boldsymbol{w}}$ we can predict the target computing $h_{\boldsymbol{w}}(x_i)$. To evaluate how good is the prediction we compute the error $(h_{\boldsymbol{w}}(x_i) - y_i)^2$.

$(h_{\boldsymbol{w}}(x_i) - y_i)^2 \geq 0$ and $(h_{\boldsymbol{w}}(x_i) - y_i)^2 = 0$ if and only if $h_{\boldsymbol{w}}(x_i) = y_i$. The **mean squared error** (MSE) is:

$$E(\boldsymbol{w}) := \frac{1}{N} \sum_{i=1}^{N} (h_{\boldsymbol{w}}(x_i) - y_i)^2.$$

To find the best model we minimize the training error, hence in this case the MSE.

$$\boldsymbol{w} \in \operatorname*{arg\,min}_{\tilde{\boldsymbol{w}} \in \mathbb{R}^2} E(\tilde{\boldsymbol{w}}).$$

n-dimensional LR

Dataset samples.

Previous case: $x \in \mathbb{R}, y \in \mathbb{R}$.

Now: $\boldsymbol{x} \in \mathbb{R}^n, y \in \mathbb{R}$.

Notation: x_j^i is the j-th coordinate of the i-th sample.

Hypothesis.

Previous case:

$$h_w(x) = w_1 x + w_0,$$

where $w = [w_0, w_1]$.

Now:

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

where $\boldsymbol{w} = [w_0, \ldots, w_n]$ and $\tilde{\boldsymbol{x}} = [1, x_1, \ldots, x_n]$.

n-dimensional LR

MSE.

Previous case:

$$E(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N}(h_{\boldsymbol{w}}(x_i) - y_i)^2.$$

Now:

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

where

$$\mathrm{X} := \begin{bmatrix} \tilde{\boldsymbol{x}}^1 \\ \vdots \\ \tilde{\boldsymbol{x}}^N \end{bmatrix} \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}.$$

Spot the minimum - Gradient descent

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

Main idea: the gradient of a scalar field represent geometrically the direction with maximum slope. Hence, following the opposite direction of the gradient lead us to get closer to the minimum of the function.

Formally:

- Start with a random $\boldsymbol{w}^0$.
- For $j \geq 0$, update $\boldsymbol{w}^{j+1} := \boldsymbol{w}^j + \boldsymbol{d}^j$, where $\boldsymbol{d}^j$ is such that

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

Gradient descent: $\boldsymbol{d}^j = -\alpha \nabla E(\boldsymbol{w}^j)$. α is called **learning rate**.

Gradient descent - 3D visualization

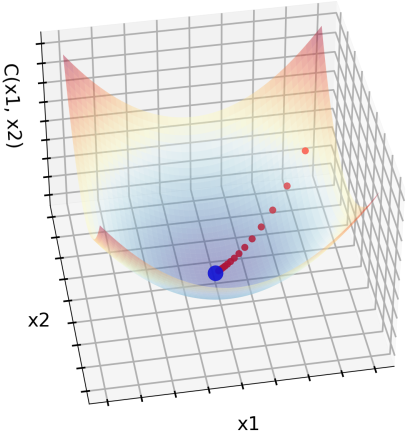


Figure: In blue the global minimum, in red the iteration points.

Gradient descent - 2D visualization

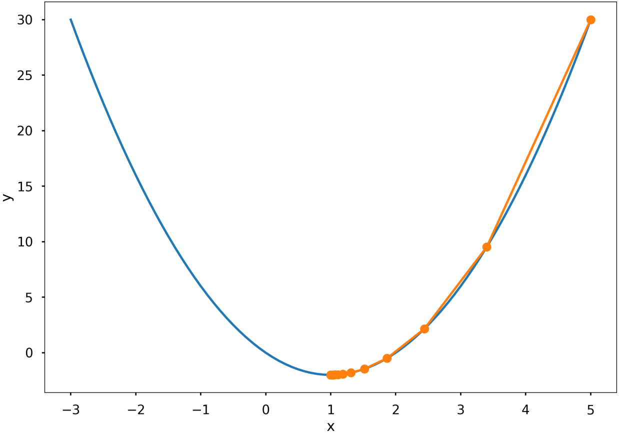


Figure: Learning rate = 0.1

Gradient descent - 2D visualization

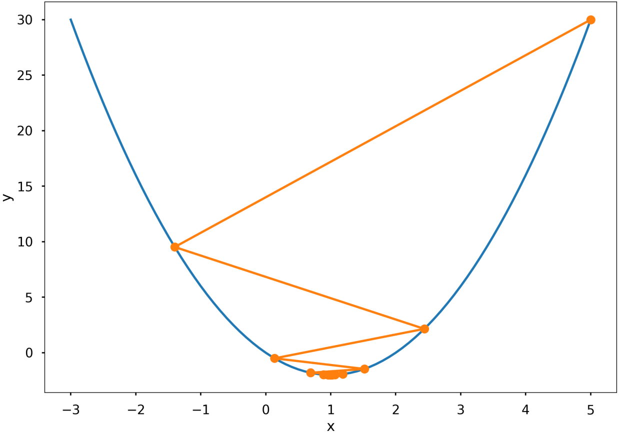


Figure: Learning rate = 0.4

Gradient descent - 2D visualization

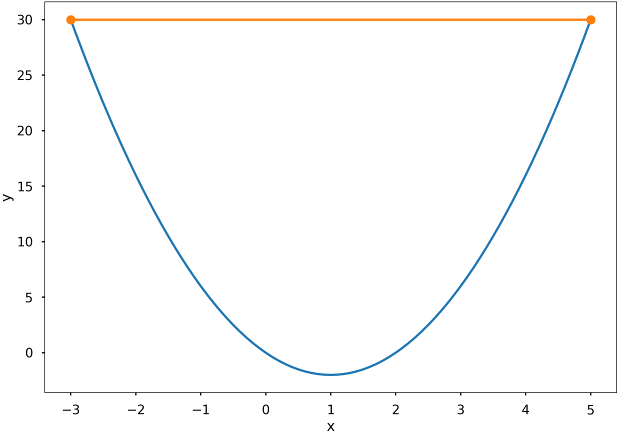


Figure: Learning rate $= 0.5$

Batch, SGD and Mini-Batch - Intuition

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

The classical gradient descent update rule is called **batch version**. We update the weights vector computing the gradient of the entire cost function $E(\boldsymbol{w})$. However, in some cases computing $\nabla E(\boldsymbol{w})$ is impossible, due to the huge amount of time to compute it.

To speed-up the update rule we can imagine that $\nabla E_i(\boldsymbol{w})$ represent a primitive approximation of $\nabla E(\boldsymbol{w})$. This is the idea behind the so called **Stochastic Gradient Descent** (SGD) or **online version**.

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

Batch, SGD and Mini-Batch - Formal

Batch

- Start with a random $\boldsymbol{w}^0$.
- For $j \geq 0$, update $\boldsymbol{w}^{j+1} := \boldsymbol{w}^j - \alpha \nabla E(\boldsymbol{w}^j)$.

Stochastic Gradient Descent (SGD or online)

- Start with a random $\boldsymbol{w}^0$.
- For $j \geq 0$ and for each pattern $1 \leq i \leq N$ update $\boldsymbol{w}^{j+1} := \boldsymbol{w}^j - \alpha \nabla E_i(\boldsymbol{w}^j)$.

Mini-Batch. Fix an integer $1 \leq \text{mb} \leq N$(mini-batch size).

- Start with a random $\boldsymbol{w}^0$.
- For $j \geq 0$ and for each pattern $0 \leq i < \frac{N}{\text{mb}}$ update

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

Tips and Tricks - How to choose?

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

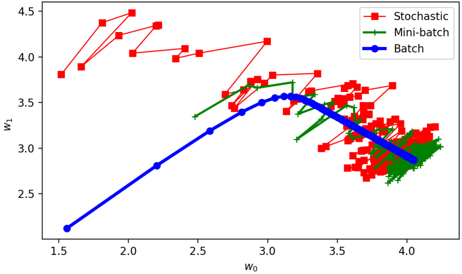


Figure: Batch vs SGD vs Mini-batch

Gradient descent and normal equation for LR

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

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

Normal equation ($\Longleftrightarrow$ holds if $\mathrm{X}^T\mathrm{X}$ is invertible):

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

Gradient descent main iteration for LR:

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

Normal equation vs gradient descent

Normal equation:

- No hyperparameter (explicit solution).
- No need to iterate.
- $\mathcal{O}(n^3)$, hence slow when n is large.

Gradient descent:

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

Tips and Tricks - Standardization

General (not only for LR): features must be on a similar scale!

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

Common techniques:

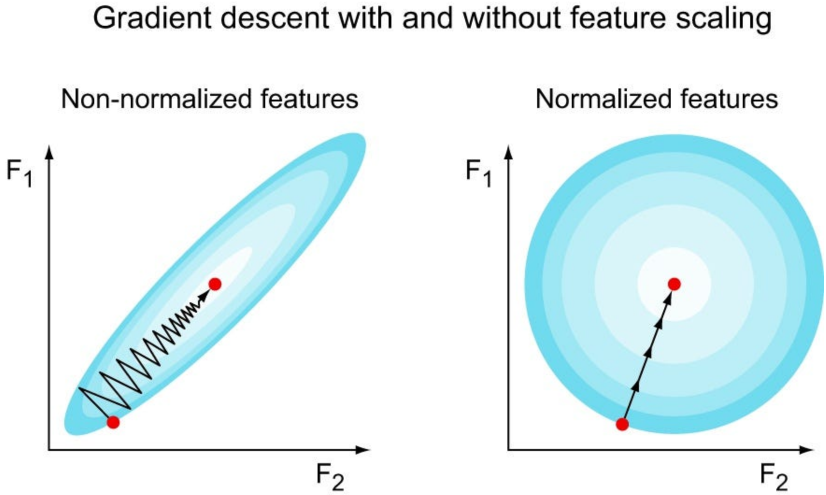
- **Feature scaling**. Compute the max $\boldsymbol{M}$ and the min $\boldsymbol{m}$ data value. Then normalize each feature as follows

$$\boldsymbol{x}^i_{\text{norm}} = \frac{\boldsymbol{x}^i - \boldsymbol{m}}{\boldsymbol{M} - \boldsymbol{m}}$$

- **Mean normalization**. Compute mean μ and standard deviation σ of the data. Then normalize each feature as follows

$$\boldsymbol{x}^i_{\text{norm}} = \frac{\boldsymbol{x}^i - \boldsymbol{\mu}}{\boldsymbol{\sigma}}$$

Tips and Tricks - Standardization



Gradient descent with and without feature scaling

Non-normalized features

Normalized features

Tips and Tricks - Invertibility of $\mathrm{X}^T\mathrm{X}$

What happens when $\mathrm{X}^T\mathrm{X}$ is not invertible?
Invertibility of $\mathrm{X}^T\mathrm{X} =$ column of X linearly independent (preprocessing information)

If a column is linearly dependent to the other then a feature is correlated with others (redundant feature).

Solution: discard that feature. The information carried by that feature is contained in some of the others.

Polynomial regression (PR)

LR corresponds to linear hypothesis, i.e. of the form

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$$

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

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

More in general: linear basis expansion (LBE)

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

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

Underfitting and overfitting - main intuition

Imagine that you have to prepare an exam.

Doing only a few exercises lead a poor perfomance both on homeworks and on the exam exercises. This is called *underfitting*: you have a bad performance on the exam because you did not trained enough.

Moreover, brutally memorize all the homework lead to a perfect score on homeworks (trivially) but probably a bad score on the exam exercises. This is called *overfitting*: you have a bad performance on the exam because you did not captured the true essence of your homework, you have also memorize their "noise" (homework pecularities).

Underfitting and overfitting - another example

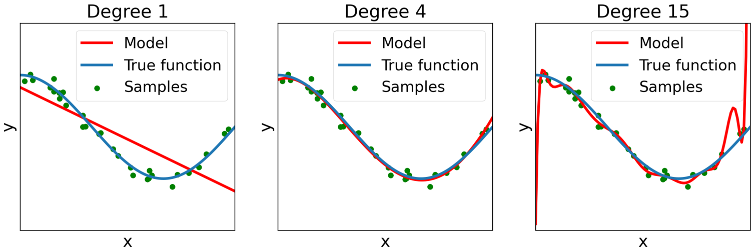


Figure: Underfitting (degree 1), good fitting (degree 4), overfitting (degree 15).

How to counter overfitting

Overfitting can be countered in many ways.

- Validation, the true core of Machine Learning.
- Early stopping.
- Ensembling.
- Regularization.

Now we focus on regularization and the general intuition behind it, lately in this course we are going to address the other points.

Tikhonov regularization

Overfitting phenomenon is highly correlated with complex models. To avoid complex models, the idea is to penalize models with large weights. One way to do it is to consider the following cost function

$$E_r(\boldsymbol{w}) = E(\boldsymbol{w}) + \underbrace{\lambda||\boldsymbol{w}||^2}_{R_\lambda(\boldsymbol{w})}.$$

R_λ is called *Tikhonov regularization* (or L^2 regularization). More in general, we will call *regularization term* the term added to the error function involving the type of regularization chosen.

$\lambda > 0$ is an hyperparameter that must be chosen in the model selection phase.

LR with Tikhonov regularization

New cost function:

$$E_r(\boldsymbol{w}) = E(\boldsymbol{w}) + \lambda ||\boldsymbol{w}||^2.$$

Gradient of the cost function:

$$\nabla E_r(\boldsymbol{w}) = \nabla E(\boldsymbol{w}) + 2\lambda \boldsymbol{w} = 2(\frac{1}{N}\mathrm{X}^T(\mathrm{X}\boldsymbol{w} - \boldsymbol{y}) + \lambda \boldsymbol{w})$$

Normal equation:

$$\boldsymbol{w} = (\mathrm{X}^T\mathrm{X} + \lambda \mathrm{I})^{-1}\mathrm{X}^T\boldsymbol{y}$$

Note that in this case $\mathrm{X}^T\mathrm{X} + \lambda\mathrm{I}$ is *always* invertible (why? :))

Most common regularization techniques

- Tikhonov regularization: $R_\lambda(\boldsymbol{w}) = \lambda||w||_2$. Tends to bring all the weights to small values.
- Lasso: $R_\lambda(\boldsymbol{w}) = \lambda||w||_1$. Tends to bring some weights to 0 (feature selection).
- Elastic net: $R_\lambda(\boldsymbol{w}) = \lambda_1||w||_1 + \lambda_2||w||_2$.