Syllabus

Fri. 21.10.	(1)	0. Introduction
		A. Supervised Learning: Linear Models & Fundamentals
Fri. 27.10.	(2)	A.1 Linear Regression
Fri. 3.11.	(3)	A.2 Linear Classification
Fri. 10.11.	(4)	A.3 Regularization
Fri. 17.11.	(5)	A.4 High-dimensional Data
		B. Supervised Learning: Nonlinear Models
Fri. 24.11.	(6)	B.1 Nearest-Neighbor Models
Fri. 1.12.	(7)	B.2 Neural Networks
Fri. 8.12.	(8)	B.3 Decision Trees
Fri. 15.12.	(9)	B.4 Support Vector Machines
Fri. 12.1.	(10)	B.5 A First Look at Bayesian and Markov Networks
		C. Unsupervised Learning
Fri. 19.1.	(11)	C.1 Clustering
Fri. 26.1.	(12)	C.2 Dimensionality Reduction
Fri. 2.2.	(13)	C.3 Frequent Pattern Mining

Outline

1. Linear Regression via Normal Equations
2. Minimizing a Function via Gradient Descent
3. Learning Linear Regression Models via Gradient Descent
4. Case Weights

Outline

The Simple Linear Regression Problem

Given

- a set $\mathcal{D}^{\text{train}} := \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\} \subseteq \mathbb{R} \times \mathbb{R}$ called **training data**,

compute the **parameters** $(\hat{\beta}_0, \hat{\beta}_1)$ of a linear **regression function**

$$\hat{y}(x) := \hat{\beta}_0 + \hat{\beta}_1 x$$

s.t. for a set $\mathcal{D}^{\text{test}} \subseteq \mathbb{R} \times \mathbb{R}$ called **test set** the **test error**

$$\text{err}(\hat{y}; \mathcal{D}^{\text{test}}) := \frac{1}{|D^{\text{test}}|} \sum_{(x,y) \in \mathcal{D}^{\text{test}}} (y - \hat{y}(x))^2$$

is minimal.

Note: $\mathcal{D}^{\text{test}}$ has (i) to be from the same data generating process and (ii) not to be available during training.

The (Multiple) Linear Regression Problem

Given

- a set $\mathcal{D}^{\text{train}} := \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\} \subseteq \mathbb{R}^M \times \mathbb{R}$ called **training data**,

compute the **parameters** $(\hat\beta_0, \hat\beta_1, \ldots, \hat\beta_M)$ of a linear **regression function**

$$\hat y(x) := \hat\beta_0 + \hat\beta_1 x_1 + \ldots + \hat\beta_M x_M$$

s.t. for a set $\mathcal{D}^{\text{test}} \subseteq \mathbb{R}^M \times \mathbb{R}$ called **test set** the **test error**

$$\text{err}(\hat y; \mathcal{D}^{\text{test}}) := \frac{1}{|D^{\text{test}}|} \sum_{(x,y)\in\mathcal{D}^{\text{test}}} (y - \hat y(x))^2$$

is minimal.

Note: $\mathcal{D}^{\text{test}}$ has (i) to be from the same data generating process and (ii) not to be available during training.

Several predictors

Several predictor variables $x_{.,1}, x_{.,2}, \ldots, x_{.,M}$:

$$\begin{aligned}\hat y =& \hat\beta_0 + \hat\beta_1 x_{.,1} + \hat\beta_2 x_{.,2} + \cdots \hat\beta_M x_{.,M} \\ =& \beta_0 + \sum_{m=1}^{M} \hat\beta_m x_{.,m}\end{aligned}$$

with $M+1$ parameters $\hat\beta_0, \hat\beta_1, \ldots, \hat\beta_M$.

Linear form

Several predictor variables $x_{.,1}, x_{.,2}, \ldots, x_{.,M}$:

$$\hat{y} = \hat{\beta}_0 + \sum_{m=1}^{M} \hat{\beta}_m x_{.,m}$$
$$= \langle \hat{\beta}, x_. \rangle$$

where

$$\hat{\beta} := \begin{pmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_M \end{pmatrix}, \quad x_. := \begin{pmatrix} 1 \\ x_{.,1} \\ \vdots \\ x_{.,M} \end{pmatrix},$$

Thus, the intercept is handled like any other parameter, for the artificial constant predictor $x_{.,0} \equiv 1$.

Simultaneous equations for the whole dataset

For the whole dataset $\mathcal{D}^{\text{train}} := \{(x_1, y_1), \ldots, (x_N, y_N)\}$:

$$y \approx \hat{y} := X\hat{\beta}$$

where

$$y := \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}, \hat{y} := \begin{pmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_N \end{pmatrix}, \quad X := \begin{pmatrix} x_1 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} x_{1,1} & x_{1,2} & \ldots & x_{1,M} \\ \vdots & \vdots & \vdots & \vdots \\ x_{N,1} & x_{N,2} & \ldots & x_{N,M} \end{pmatrix}$$

Least squares estimates

Least squares estimates $\hat{\beta}$ minimize

$$\text{RSS}(\hat{\beta}, \mathcal{D}^{\text{train}}) := \sum_{n=1}^{N} (y_n - \hat{y}_n)^2 = ||y - \hat{y}||^2 = ||y - X\hat{\beta}||^2$$

$$\hat{\beta} := \arg\min_{\hat{\beta} \in \mathbb{R}^M} ||y - X\hat{\beta}||^2$$

The least squares estimates $\hat{\beta}$ can be computed analytically via **normal equations**

$$X^T X \hat{\beta} = X^T y$$

Proof:
$$||y - X\hat{\beta}||^2 = \langle y - X\hat{\beta}, y - X\hat{\beta} \rangle$$

$$\frac{\partial(\ldots)}{\partial\hat{\beta}} = 2\langle -X, y - X\hat{\beta}\rangle = -2(X^T y - X^T X \hat{\beta}) \stackrel{!}{=} 0$$

How to compute least squares estimates $\hat{\beta}$

Solve the $M \times M$ system of linear equations

$$X^T X \hat{\beta} = X^T y$$

i.e., $Ax = b$ (with $A := X^T X, b = X^T y, x = \hat{\beta}$).

There are several numerical methods available:

1. Gaussian elimination
2. Cholesky decomposition
3. QR decomposition

Learn Linear Regression via Normal Equations

procedure $\text{LEARN-LINREG-NORMEQ}(\mathcal{D}^{\text{train}} := \{(x_1, y_1), \ldots, (x_N, y_N)\})$

$\quad X := (x_1, x_2, \ldots, x_N)^T$

$\quad y := (y_1, y_2, \ldots, y_N)^T$

$\quad A := X^T X$

$\quad b := X^T y$

$\quad \hat{\beta} := \text{SOLVE-SLE}(A, b)$

$\quad$ **return** $\hat{\beta}$

Example

Given is the following data:

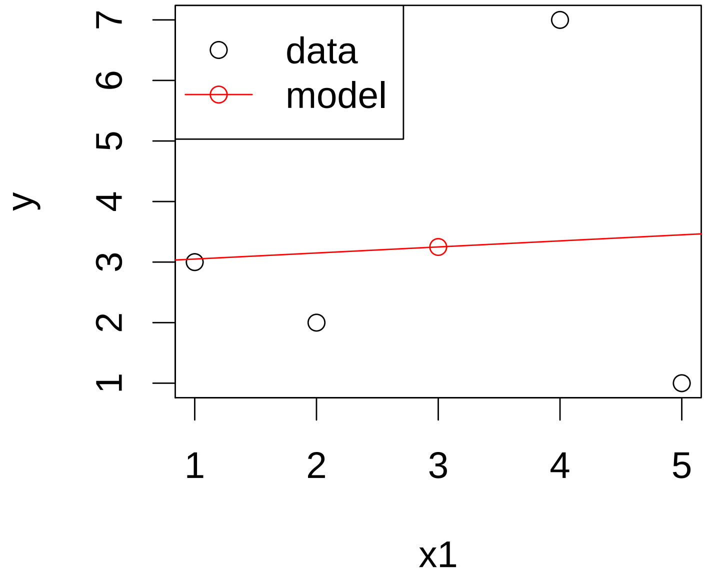
x_1	x_2	y
1	2	3
2	3	2
4	1	7
5	5	1

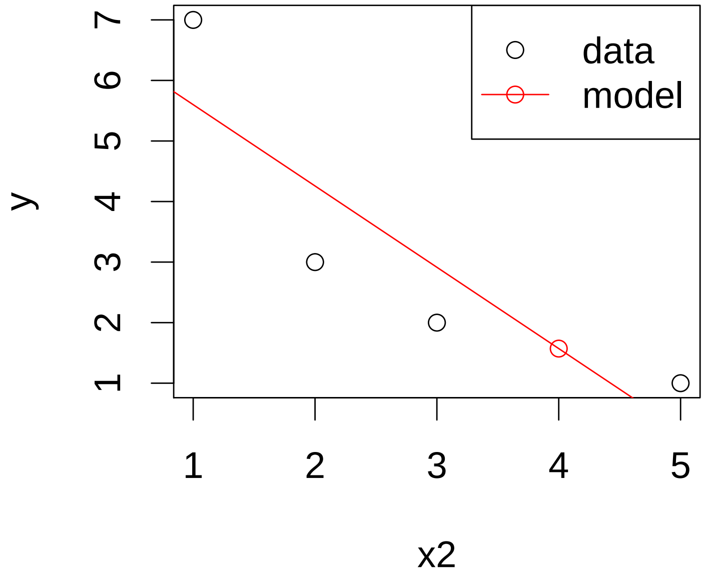
Predict a y value for $x_1 = 3, x_2 = 4$.

Example / Simple Regression Models for Comparison

$$\begin{aligned}\hat{y} =& \hat{\beta}_0 + \hat{\beta}_1 x_1 \\ =& 2.95 + 0.1 x_1\end{aligned}$$

$$\begin{aligned}\hat{y} =& \hat{\beta}_0 + \hat{\beta}_2 x_2 \\ =& 6.943 - 1.343 x_2\end{aligned}$$





$$\hat{y}(x_1 = 3) = 3.25$$

$$\text{RSS} = 20.65, \quad \text{RMSE} = 4.02$$

$$\hat{y}(x_2 = 4) = 1.571$$

$$\text{RSS} = 4.97, \quad \text{RMSE} = 2.13$$

Example

Now fit

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2$$

to the data:

x_1	x_2	y
1	2	3
2	3	2
4	1	7
5	5	1

$$X = \begin{pmatrix} 1 & 1 & 2 \\ 1 & 2 & 3 \\ 1 & 4 & 1 \\ 1 & 5 & 5 \end{pmatrix}, \quad y = \begin{pmatrix} 3 \\ 2 \\ 7 \\ 1 \end{pmatrix}$$

$$X^T X = \begin{pmatrix} 4 & 12 & 11 \\ 12 & 46 & 37 \\ 11 & 37 & 39 \end{pmatrix}, \quad X^T y = \begin{pmatrix} 13 \\ 40 \\ 24 \end{pmatrix}$$

Example

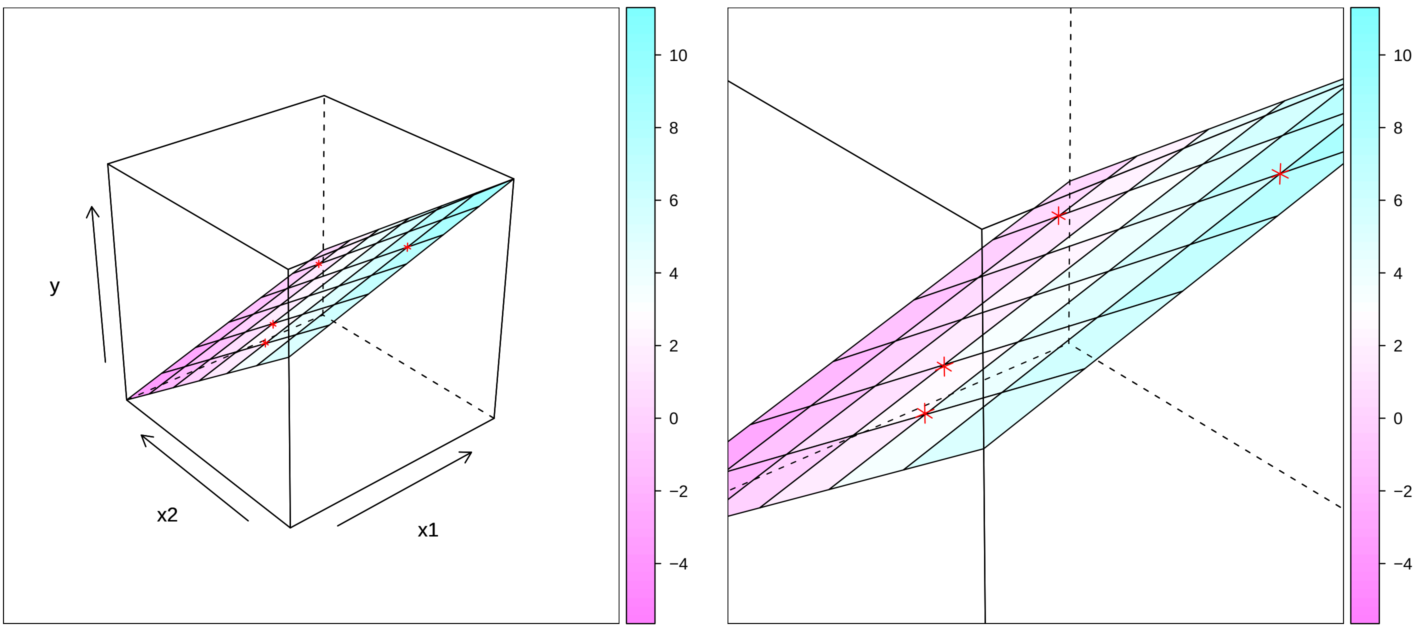
$$\left(\begin{array}{ccc|c} 4 & 12 & 11 & 13 \\ 12 & 46 & 37 & 40 \\ 11 & 37 & 39 & 24 \end{array}\right) \sim \left(\begin{array}{ccc|c} 4 & 12 & 11 & 13 \\ 0 & 10 & 4 & 1 \\ 0 & 16 & 35 & -47 \end{array}\right) \sim \left(\begin{array}{ccc|c} 4 & 12 & 11 & 13 \\ 0 & 10 & 4 & 1 \\ 0 & 0 & 143 & -243 \end{array}\right)$$

$$\sim \left(\begin{array}{ccc|c} 4 & 12 & 11 & 13 \\ 0 & 1430 & 0 & 1115 \\ 0 & 0 & 143 & -243 \end{array}\right) \sim \left(\begin{array}{ccc|c} 286 & 0 & 0 & 1597 \\ 0 & 1430 & 0 & 1115 \\ 0 & 0 & 143 & -243 \end{array}\right)$$

i.e.,

$$\hat{\beta} = \begin{pmatrix} 1597/286 \\ 1115/1430 \\ -243/143 \end{pmatrix} \approx \begin{pmatrix} 5.583 \\ 0.779 \\ -1.699 \end{pmatrix}$$

Example



$$\hat{y}(x_1 = 3, x_2 = 4) = 1.126$$

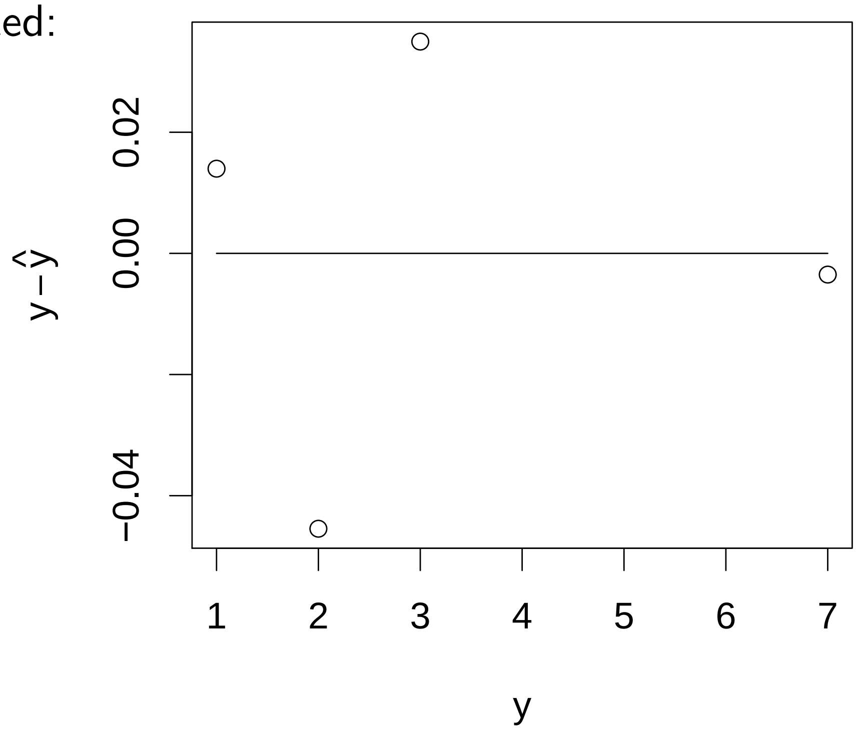
$$\text{RSS} = 0.0035, \quad \text{RMSE} = 0.58$$

Example / Visualization of Model Fit

To visually assess the model fit, a **scatter plot**

$$\text{residuals } \hat{\epsilon} := y - \hat{y} \text{ vs. true values } y$$

can be plotted:



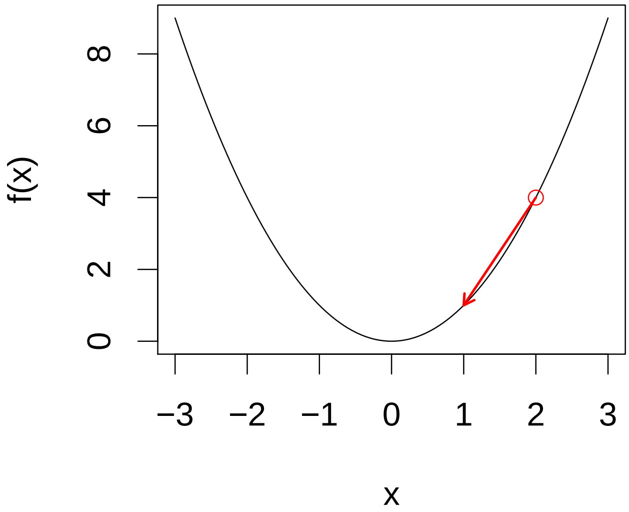
Outline

Gradient Descent

Given a function $f : \mathbb{R}^N \rightarrow \mathbb{R}$, find x with minimal $f(x)$.

Idea: start from a random x_0 and then improve step by step, i.e., choose x_{i+1} with

$$f(x_{i+1}) \leq f(x_i)$$



Choose the negative gradient $-\frac{\partial f}{\partial x}(x_i)$ as direction for descent, i.e.,

$$x_{i+1} - x_i = -\alpha_i \cdot \frac{\partial f}{\partial x}(x_i)$$

with a suitable step length $\alpha_i > 0$.

Gradient Descent

procedure $\textsc{minimize-GD-fsl}(f : \mathbb{R}^N \rightarrow \mathbb{R}, x_0 \in \mathbb{R}^N, \alpha \in \mathbb{R}, i_{\max} \in \mathbb{N}, \epsilon \in \mathbb{R}^+)$

    for $i = 1, \ldots, i_{\max}$ **do**

        $x_i := x_{i-1} - \alpha \cdot \frac{\partial f}{\partial x}(x_{i-1})$

        if $f(x_{i-1}) - f(x_i) < \epsilon$ **then**

            return x_i

    error "not converged in $i_{\max}$ iterations"

x_0	start value
α	(fixed) step length / learning rate
$i_{\max}$	maximal number of iterations
ϵ	minimum stepwise improvement

Example

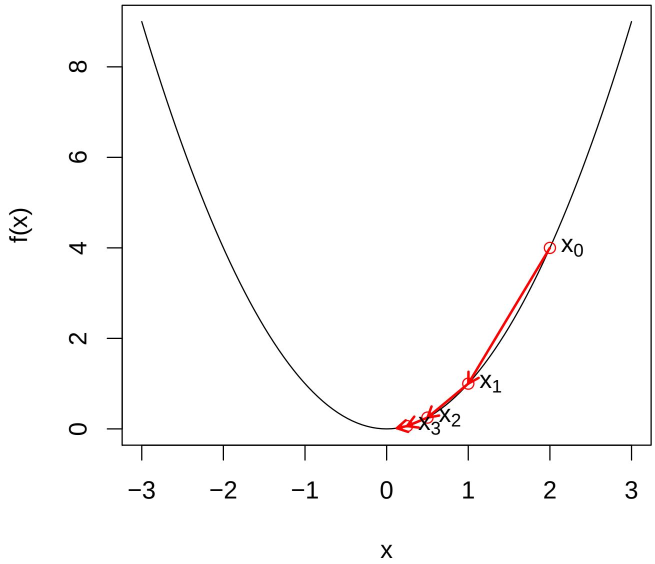
$$f(x) := x^2, \quad \frac{\partial f}{\partial x}(x) = 2x, \quad x_0 := 2, \quad \alpha_i :\equiv 0.25$$

Then we compute iteratively:

i	x_i	$\frac{\partial f}{\partial x}(x_i)$	x_{i+1}
0	2	4	1
1	1	2	0.5
2	0.5	1	0.25
3	0.25	$\vdots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

using

$$x_{i+1} = x_i - \alpha_n \cdot \frac{\partial f}{\partial x}(x_i)$$

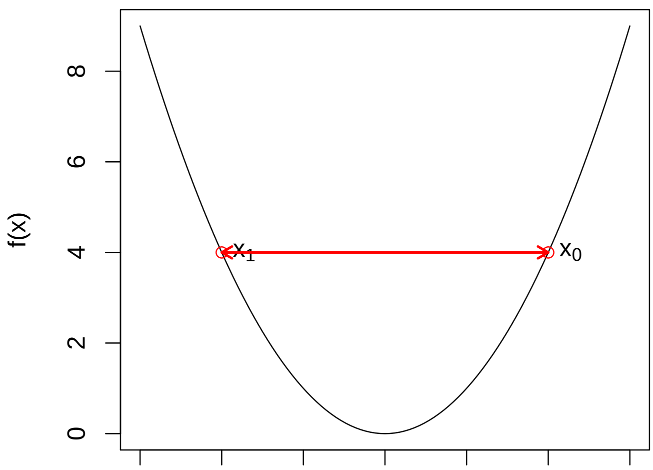


Step Length

Why do we need a step length? Can we set $\alpha_n \equiv 1$?

The negative gradient gives a direction of descent only in an infinitesimal neighborhood of x_n.

Thus, the step length may be too large, and the function value of the next point does not decrease.



Step Length

There are many different strategies to adapt the step length s.t.

1. the function value actually decreases and
2. the step length becomes not too small
   (and thus convergence slow)

Armijo-Principle:

$$\alpha_n := \max\{\alpha \in \{2^{-j} \mid j \in \mathbb{N}_0\} \mid f(x_n - \alpha \frac{\partial f}{\partial x}(x_n)) \leq f(x_n) - \alpha\delta\langle \frac{\partial f}{\partial x}(x_n), \frac{\partial f}{\partial x}(x_n)\rangle \}$$

with $\delta \in (0, 1)$.

Armijo Step Length

procedure STEPLENGTH-ARMIJO$(f : \mathbb{R}^N \to \mathbb{R}, x \in \mathbb{R}^N, d \in \mathbb{R}^N, \delta \in (0, 1))$
  $\alpha := 1$
  while $f(x) - f(x + \alpha d) < \alpha\delta d^T d$ **do**
    $\alpha = \alpha/2$
  return α

x last position

d descend direction

δ minimum steepness ($\delta \approx 0$: any step will do)

Gradient Descent

procedure MINIMIZE-GD($f : \mathbb{R}^N \rightarrow \mathbb{R}, x_0 \in \mathbb{R}^N, \alpha, i_{\max} \in \mathbb{N}, \epsilon \in \mathbb{R}^+$)
  for $i = 1, \ldots, i_{\max}$ **do**
    $d := -\frac{\partial f}{\partial x}(x_{i-1})$
    $\alpha_i := \alpha(f, x_{i-1}, d)$
    $x_i := x_{i-1} + \alpha_i \cdot d$
    if $f(x_{i-1}) - f(x_i) < \epsilon$ **then**
      return x_i
  error "not converged in $i_{\max}$ iterations"

- x_0 start value
- α step length function, e.g., STEPLENGTH-ARMIJO (with fixed δ).
- $i_{\max}$ maximal number of iterations
- ϵ minimum stepwise improvement

Bold Driver Step Length [Bat89]

A variant of the Armijo principle with memory:

procedure STEPLENGTH-BOLDDRIVER($f : \mathbb{R}^N \rightarrow \mathbb{R}, x \in \mathbb{R}^N, d \in \mathbb{R}^N, \alpha^{\text{old}}, \alpha^+, \alpha^- \in (0,1)$)
  $\alpha := \alpha^{\text{old}} \alpha^+$
  while $f(x) - f(x + \alpha d) \leq 0$ **do**
    $\alpha = \alpha \alpha^-$
  return α

- α^{old} last step length
- α^+ step length increase factor, e.g., 1.1.
- α^- step length decrease factor, e.g., 0.5.

Simple Step Length Control in Machine Learning

- The Armijo and Bold Driver step lengths evaluate the objective function (including the loss) several times, and thus often are too costly and not used.
  - But useful for debugging as they guarantee decrease in f.

- Constant step lengths $\alpha \in (0, 1)$ are frequently used.
  - If chosen (too) small, the learning algorithm becomes slow, but usually still converges.
  - The step length becomes a hyperparameter that has to be searched.

- Regimes of shrinking step lengths are used:

$$\alpha_i := \alpha_{i-1}\gamma, \quad \gamma \in (0, 1) \text{ not too far from } 1$$

  - If the initial step length α_0 is too large, later iterations will fix it.
  - If γ is too small, GD may get stuck before convergence.

How Many Minima can a Function have?

- In general, a function f can have several different **local minima** i.e., points x with $\frac{\partial f}{\partial x}(x) = 0$.

- GD will find a random one
(with small step lengths, usually one close to the starting point; **local optimization method**).

Convexity

- A function $f : \mathbb{R}^N \rightarrow \mathbb{R}$ is called **convex** if

$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2), \quad \forall x_1, x_2 \in \mathbb{R}^N, t \in [0,1]$$

- for a convex function, all local minima have the same function value (**global minimum**)

- **2nd-order criterion for convexity**: A two-times differentiable function is convex if its Hessian is positive semidefinite, i.e.,

$$x^T \left(\frac{\partial^2 f}{\partial x_i \partial x_j} \right)_{i=1,\ldots,N, j=1,\ldots,N} x \geq 0 \quad \forall x \in \mathbb{R}^N$$

- For any matrix $A \in \mathbb{R}^{N \times M}$, the matrix $A^T A$ is positive semidefinite.

Outline

Sparse Predictors

Many problems have predictors $x \in R^M$ that are

- **high-dimensional**: M is large, and
- **sparse**: most x_m are zero.

For example, **text regression**:

- task: predict the rating of a customer review.
- predictors: a text about a product — a sequence of words.
  - can be represented as vector via **bag of words**: x_m encodes the frequency of word m in a given text.
  - dimensions 30,000-60,000 for English texts
  - in short texts as reviews with a couple of hundred words, maximally a couple of hundred dimensions are non-zero.
- target: the customers rating of the product — a (real) number.

Sparse Predictors — Dense Normal Equations

- Recall, the normal equations

$$X^T X \hat{\beta} = X^T y$$

  have dimensions $M \times M$.

- Even if X is sparse, generally $X^T X$ will be rather dense.

$$(X^T X)_{m,l} = X_{.,m}^T X_{.,l}$$

  - For text regression, $(X^T X)_{m,l}$ will be non-zero for every pair of words m, l that co-occurs in any text.

- Even worse, even if $A := X^T X$ is sparse, standard methods to solve linear systems (such as Gaussian elimination, LR decomposition etc.) do not take advantage.

Learn Linear Regression via Loss Minimization

Alternatively to learning a linear regression model via solving the linear normal equation system one can minimize the loss directly:

$$\begin{aligned} f(\hat{\beta}) &:= \hat{\beta}^T X^T X \hat{\beta} - 2y^T X \hat{\beta} + y^T y \\ &= (y - X\hat{\beta})^T (y - X\hat{\beta}) \\ \frac{\partial f}{\partial \hat{\beta}}(\hat{\beta}) &= -2(X^T y - X^T X \hat{\beta}) \\ &= -2X^T(y - X\hat{\beta}) \end{aligned}$$

When computing f and $\frac{\partial f}{\partial \hat{\beta}}$,

- avoid computing (dense) $X^T X$.
- always compute (sparse) X times a (dense) vector.

Objective Function and Gradient as Sums over Instances

$$\begin{aligned} f(\hat{\beta}) &:= (y - X\hat{\beta})^T (y - X\hat{\beta})^T \\ &= \sum_{n=1}^{N} (y_n - x_n^T \hat{\beta})^2 \\ &= \sum_{n=1}^{N} \epsilon_n^2, \qquad \epsilon_n := y_n - x_n^T \hat{\beta} \\ \frac{\partial f}{\partial \hat{\beta}}(\hat{\beta}) &= -2X^T(y - X\hat{\beta}) \\ &= -2 \sum_{n=1}^{N} (y_n - x_n^T \hat{\beta}) x_n \\ &= -2 \sum_{n=1}^{N} \epsilon_n x_n \end{aligned}$$

Learn Linear Regression via Loss Minimization: GD

procedure LEARN-LINREG-GD($\mathcal{D}^{\text{train}} := \{(x_1, y_1), \ldots, (x_N, y_N)\}, \alpha, i_{\max} \in \mathbb{N}, \epsilon \in \mathbb{R}^+$)

$X := (x_1, x_2, \ldots, x_N)^T$

$y := (y_1, y_2, \ldots, y_N)^T$

$\hat{\beta}_0 := (0, \ldots, 0)$

$\hat{\beta} :=$ MINIMIZE-GD($f(\hat{\beta}) := (y - X\hat{\beta})^T (y - X\hat{\beta}),$ $\hat{\beta}_0, \alpha, i_{\max}, \epsilon$)

return $\hat{\beta}$

Outline

1. Linear Regression via Normal Equations
2. Minimizing a Function via Gradient Descent
3. Learning Linear Regression Models via Gradient Descent
4. Case Weights

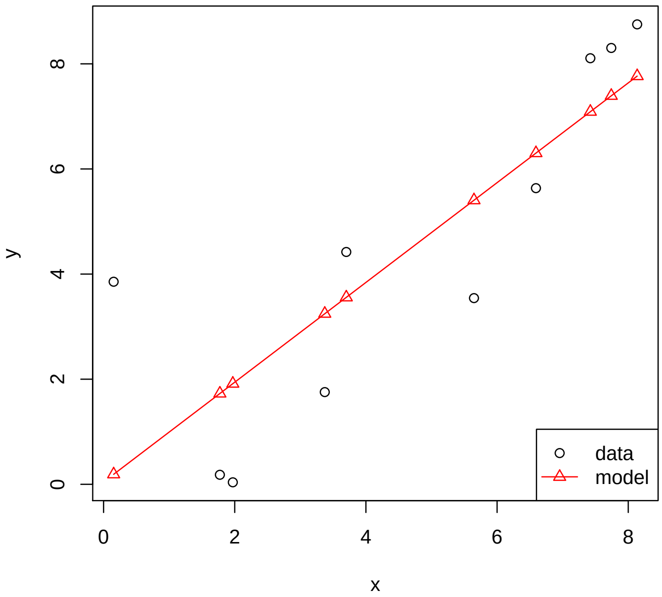
Cases of Different Importance

Sometimes different cases are of different importance, e.g., if their measurements are of different accurracy or reliability.

Example: assume the left most point is known to be measured with lower reliability.

Thus, the model does not need to fit to this point equally as well as it needs to do to the other points.

I.e., residuals of this point should get lower weight than the others.



Case Weights

In such situations, each case (x_n, y_n) is assigned a **case weight** $w_n \geq 0$:

- the higher the weight, the more important the case.
- cases with weight 0 should be treated as if they have been discarded from the data set.

Case weights can be managed as an additional pseudo-variable w in implementations.

Weighted Least Squares Estimates

Formally, one tries to minimize the **weighted residual sum of squares**

$$\sum_{n=1}^{N} w_n (y_n - \hat{y}_n)^2 = ||W^{\frac{1}{2}}(y - \hat{y})||^2$$

with

$$W := \begin{pmatrix} w_1 & & & 0 \\ & w_2 & & \\ & & \ddots & \\ 0 & & & w_n \end{pmatrix}$$

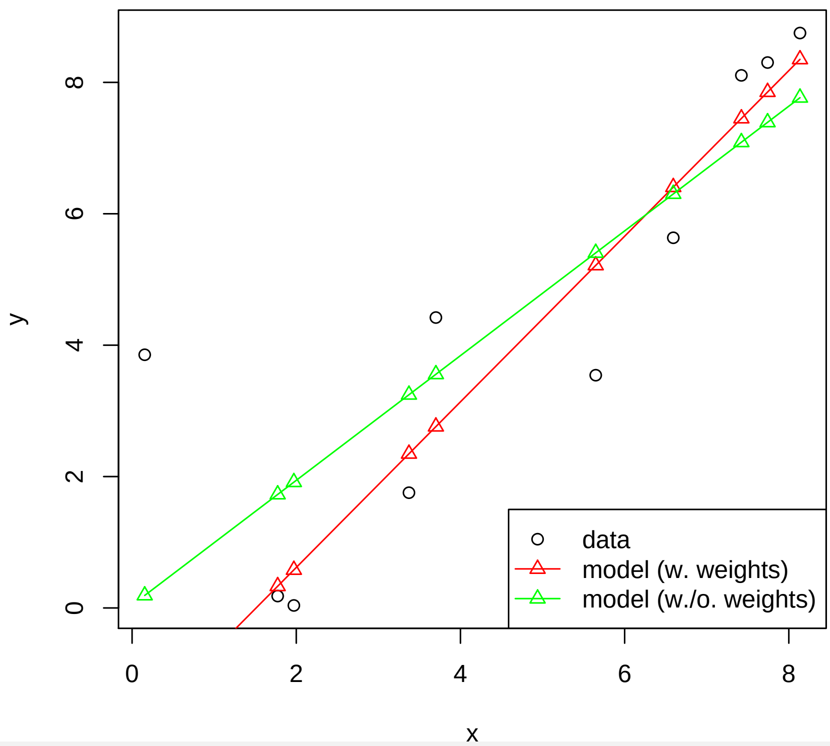
The same argument as for the unweighted case results in the **weighted least squares estimates**

$$X^T W X \hat{\beta} = X^T W y$$

Weighted Least Squares Estimates / Example

To downweight the left most point, we assign case weights as follows:

w	x	y
1	5.65	3.54
1	3.37	1.75
1	1.97	0.04
1	3.70	4.42
0.1	0.15	3.85
1	8.14	8.75
1	7.42	8.11
1	6.59	5.64
1	1.77	0.18
1	7.74	8.30



Summary

- For regression, **linear models** of type $\hat{y} = x^T \hat{\beta}$ can be used to predict a quantitative y based on several (quantitative) x.
  - A **bias term** can be modeled as additional predictor that is constant 1.
- The **ordinary least squares estimates (OLS)** are the parameters with minimal **residual sum of squares** (RSS).
- OLS estimates can be computed by solving the **normal equations** $X^T X \hat{\beta} = X^T y$ as any system of linear equations via **Gaussian elimination**.
- Alternatively, OLS estimates can be computed iteratively via **Gradient Descent**.
  - Especially for **high-dimensional, sparse predictors** GD is advantageous as it never has it never has to compute the large, dense $X^T X$.
- **Case weights** can be handled seamlessly by both methods to model different importance of cases.

Further Readings

- [JWHT13, chapter 3], [Mur12, chapter 7], [HTFF05, chapter 3].

References

Roberto Battiti.
Accelerated backpropagation learning: Two optimization methods.
Complex systems, 3(4):331–342, 1989.

Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin.
The elements of statistical learning: data mining, inference and prediction, volume 27.
Springer, 2005.

Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani.
An introduction to statistical learning.
Springer, 2013.

Kevin P. Murphy.
Machine learning: a probabilistic perspective.
The MIT Press, 2012.