Lecture 3.2
Supervised learning: Regression

Universitat de les Illes Balears
Departament de Ciències Matemàtiques i Informàtica

11752 Aprendizaje Automático
11752 Machine Learning
Máster Universitario
en Sistemas Inteligentes

Alberto ORTIZ RODRÍGUEZ

Contents

- **Task**: Learn a regression function

$$f : \mathbb{R}^L \longrightarrow \mathbb{R}$$
$$f(x) = y$$

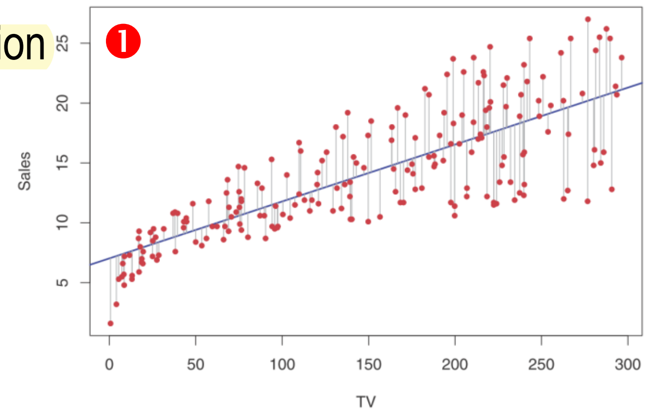
- **Goal**: Be able to predict f for any x in the domain

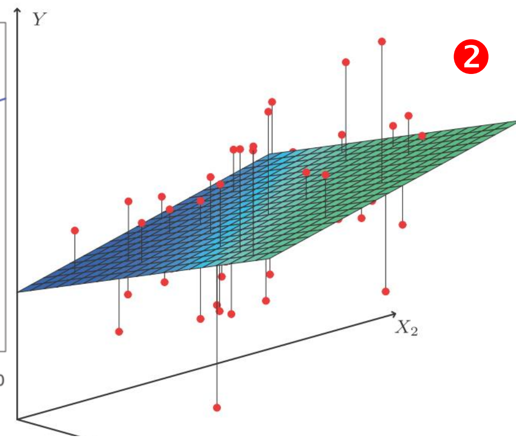
i.e. learn the **parameters** of a model

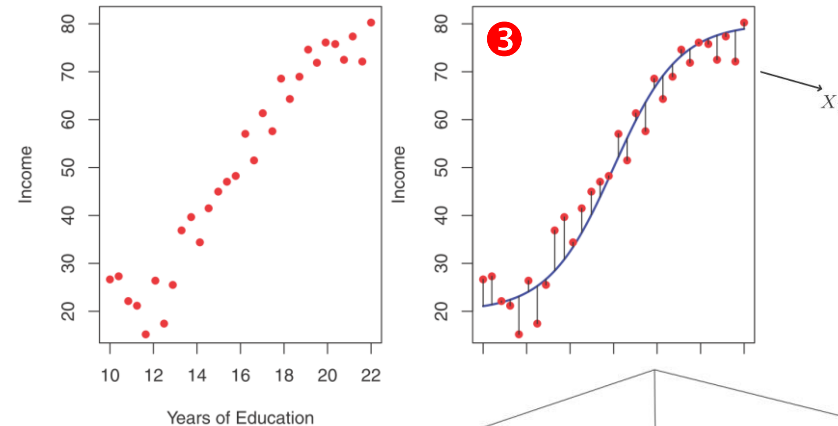
❶ $\text{sales} = \beta_0 + \beta_1 \times \text{TV}$

❷ $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$

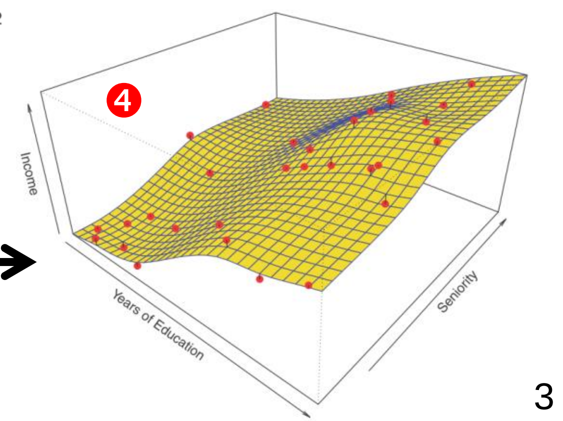
❸ $\text{income} = \gamma_0 + \dfrac{\gamma_1}{1 + e^{-(\beta_0 + \beta_1 \times \text{yoe})}}$







- A regression model is said to be **linear** if it is expressed by a linear function ❶ & ❷ or **non-linear** ❸
- Regression can also be **non-parametric** ❹



Contents

- We consider a **linear model** such as the following:

$$f(x;\theta=\{\beta_j\}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_L x_L, \text{ with } \beta_j \in \mathbb{R},\ x = (x_1,\ldots,x_L)$$
$$= \beta_0 + \sum_{j=1}^{L} \beta_j x_j$$

  – We assume our dataset consists of a collection of N pairs (x_i, y_i) with $x_i = (x_{i1}, \ldots, x_{iL})$

- We next define the **sum of squared residuals**, also known as the *residual sum of squares* (RSS), as:

$$\mathrm{RSS}(\theta) = (y_1 - \widehat{y}_1)^2 + (y_2 - \widehat{y}_2)^2 + \cdots + (y_N - \widehat{y}_N)^2, \text{ with } \widehat{y}_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_L x_{iL}$$
$$= e_1^2 + e_2^2 + \cdots + e_N^2$$

- And now we adopt a **least squares formulation** to find out θ:

$$\widehat{\theta} = \{\beta_j\} = \arg\min_{\theta} \frac{1}{2}\mathrm{RSS}(\theta) = \frac{1}{2}\sum_{i=1}^{N}(y_i - f(x_i;\theta))^2$$

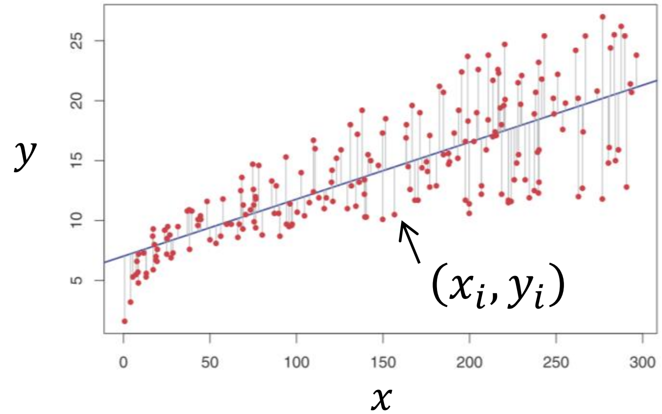
  – The term $\frac{1}{2}\sum(y_i - f(x_i))^2$ is known as the **least squares loss** (or **loss function** in general, also known as the *risk* or *cost function*)
  – Hence, we intend to find the **minimizer** θ of the loss function

Linear regression

- A simple case with **one feature** (L = 1):

$$f(x; \beta_0, \beta_1) = \beta_0 + \beta_1 x$$

  – Our data consists of pairs $(x_i, y_i), i = 1, \dots, N$



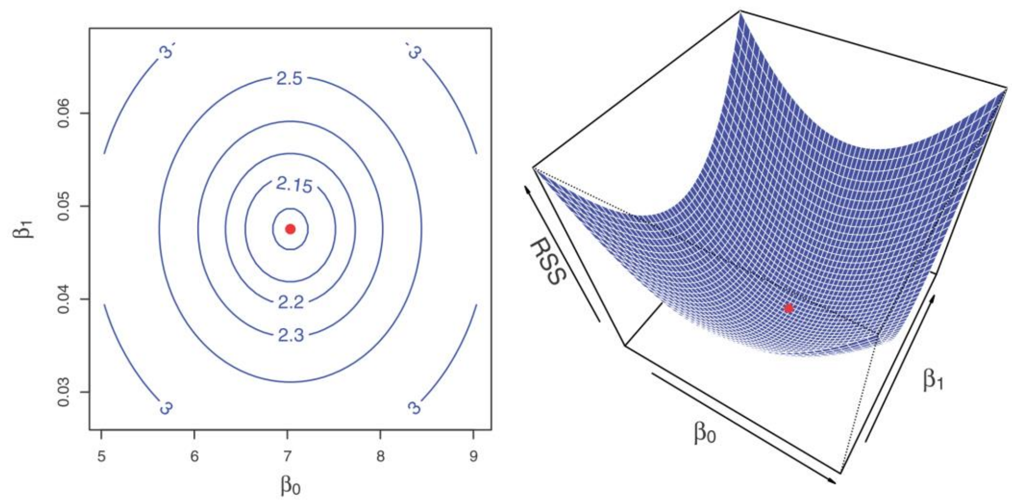
- Therefore:

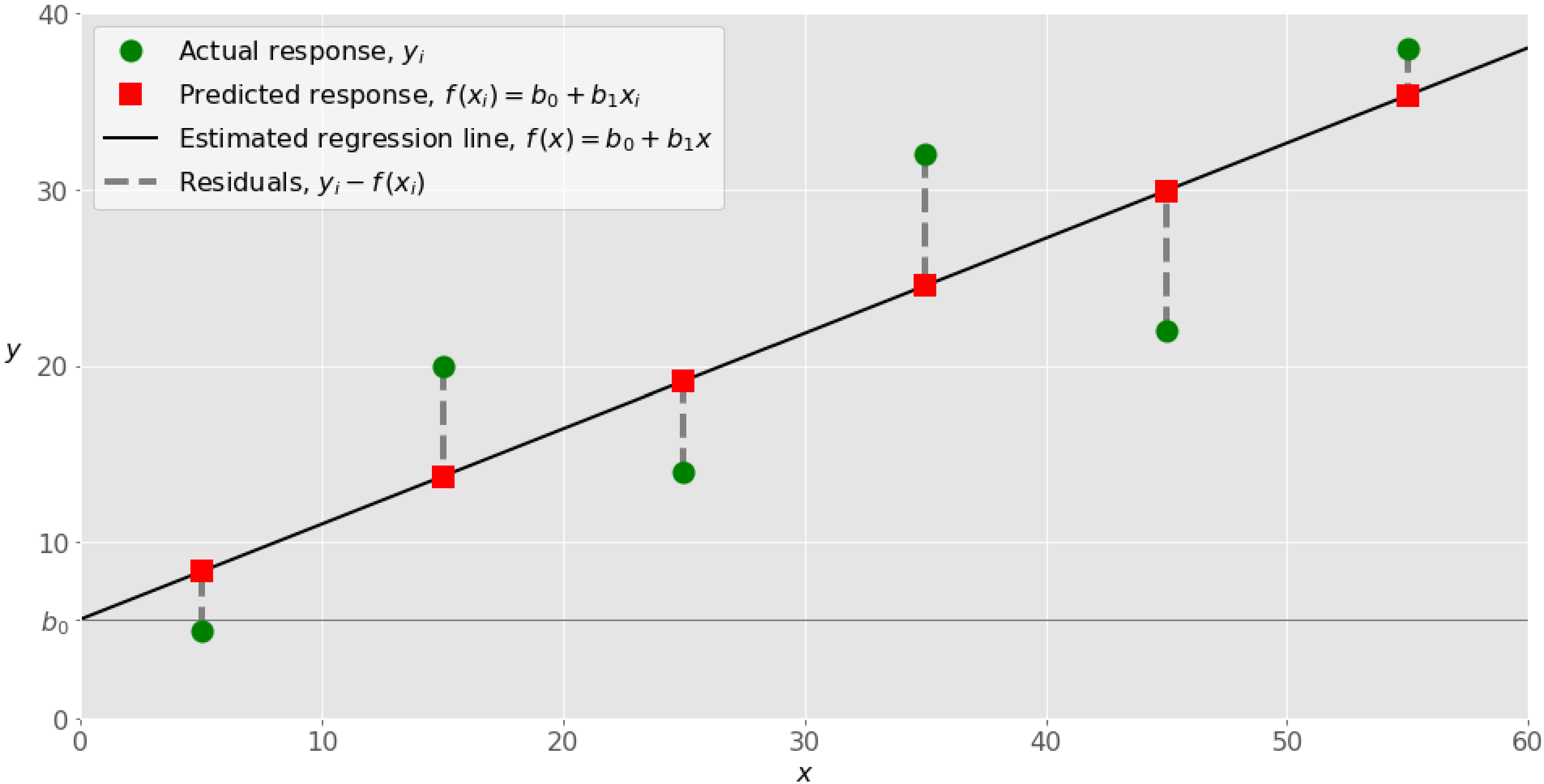
$$\text{RSS}(\theta) = (y_1 - (\beta_0 + \beta_1 x_1))^2 + (y_2 - (\beta_0 + \beta_1 x_2))^2 + \cdots + (y_N - (\beta_0 + \beta_1 x_N))^2$$

and the least squares formulation becomes into:

$$\widehat{\theta} = \{\beta_0, \beta_1\} = \underset{\beta_0, \beta_1}{\arg\min}\; L(\beta_0, \beta_1) = \frac{1}{2}\sum_{i=1}^{N}(y_i - (\beta_0 + \beta_1 x_i))^2$$

- This is an optimization problem with one single minimum at $(\hat{\beta}_0, \hat{\beta}_1)$





Actual response, y_i
Predicted response, $f(x_i) = b_0 + b_1 x_i$
Estimated regression line, $f(x) = b_0 + b_1 x$
Residuals, $y_i - f(x_i)$
40
30
20
10
b_0
0
y
0
10
20
30
40
50
60
x

- Taking derivatives and setting them equal to zero:

$$\theta = \{\beta_0, \beta_1\} = \arg\min_{\beta_0,\beta_1} \; L(\beta_0, \beta_1) = \frac{1}{2}\sum_{i=1}^{N}(y_i - (\beta_0 + \beta_1 x_i))^2$$

$$\frac{\partial L}{\partial \beta_0} = 0 \Rightarrow \sum_{i=1}^{N}(y_i - (\beta_0 + \beta_1 x_i))(-1) = 0 \Rightarrow \sum_i \beta_0 + \sum_i \beta_1 x_i = \sum_i y_i$$

$$\frac{\partial L}{\partial \beta_1} = 0 \Rightarrow \sum_{i=1}^{N}(y_i - (\beta_0 + \beta_1 x_i))(-x_i) = 0 \Rightarrow \sum_i \beta_0 x_i + \sum_i \beta_1 x_i^2 = \sum_i x_i y_i$$

$$\left.\begin{array}{ccccccc} N & \beta_0 & + & \left(\sum_i x_i\right) & \beta_1 & = & \sum_i y_i \\ \left(\sum_i x_i\right) & \beta_0 & + & \left(\sum_i x_i^2\right) & \beta_1 & = & \sum_i x_i y_i \end{array}\right\} \Rightarrow \widehat{\beta}_0 = \frac{\begin{vmatrix} \sum_i y_i & \sum_i x_i \\ \sum_i x_i y_i & \sum_i x_i^2 \end{vmatrix}}{\begin{vmatrix} N & \sum_i x_i \\ \sum_i x_i & \sum_i x_i^2 \end{vmatrix}}$$

$$\widehat{\beta}_1 = \frac{\begin{vmatrix} \sum_i N & \sum_i y_i \\ \sum_i x_i & \sum_i x_i y_i \end{vmatrix}}{\begin{vmatrix} N & \sum_i x_i \\ \sum_i x_i & \sum_i x_i^2 \end{vmatrix}}$$

$$\boxed{\widehat{\beta}_0 = \frac{\left(\sum_i y_i\right)\left(\sum_i x_i^2\right) - \left(\sum_i x_i\right)\left(\sum_i x_i y_i\right)}{N\left(\sum x_i^2\right) - \left(\sum_i x_i\right)^2}} \quad \boxed{\widehat{\beta}_1 = \frac{N\left(\sum_i x_i y_i\right) - \left(\sum_i x_i\right)\left(\sum_i y_i\right)}{N\left(\sum x_i^2\right) - \left(\sum_i x_i\right)^2}}$$

- It is even possible to estimate the reliability of these estimates through the so-called **standard errors** (SE, i.e. how far to the true value is the estimate):

$$\mathrm{SE}(\widehat{\beta}_0)^2 = \sigma_M^2 \frac{\sum_i x_i^2/N}{\sum_i (x_i - \bar{x})^2} \qquad \mathrm{SE}(\widehat{\beta}_1)^2 = \frac{\sigma_M^2}{\sum_i (x_i - \bar{x})^2}$$

  where σ_M^2 is the variance of the noise of the model ϵ, i.e. $y = \beta_0 + \beta_1 x + \epsilon$ and $\bar{x} = \sum_i x_i/N$

  σ_M is a priori unknown but can be estimated from the data by means of the **residual standard error** (RSE):

$$\widehat{\sigma}_M = \mathrm{RSE} = \sqrt{\mathrm{RSS}/(N-2)}$$

- Then, we can set e.g. **95% confidence intervals** (approximate) for the model parameters:

$$\mathrm{CI}_{95\%}(\widehat{\beta}_0) = \widehat{\beta}_0 \pm 2 \cdot \mathrm{SE}(\widehat{\beta}_0) \qquad \mathrm{CI}_{95\%}(\widehat{\beta}_1) = \widehat{\beta}_1 \pm 2 \cdot \mathrm{SE}(\widehat{\beta}_1)$$

- The RSE is actually a measure of **how well the model fits the data**:
  - RSE ↓↓ means good fit
  - RSE ↑↑ means bad fit
- The **R^2 statistic** ($\in [0,1]$) is an alternative measure of fit:
  - The closer to 1, the better is the fit
  - Also known as **coefficient of determination**

$$R^2 = 1 - \frac{\sum_i (y_i - \widehat{y}_i)^2}{\sum_i (y_i - \bar{y})^2} = 1 - \frac{\mathrm{RSS}(y, \widehat{y})}{N \cdot \mathrm{Var}(y)}$$

- **Example**. Let us consider the advertising dataset, with observed sales for a given product (as thousands of units sold) for 200 markets as a function of the budget in TV advertising (in $1K)
- Using the expressions for $\hat{\beta}_0, \hat{\beta}_1$ we obtain:

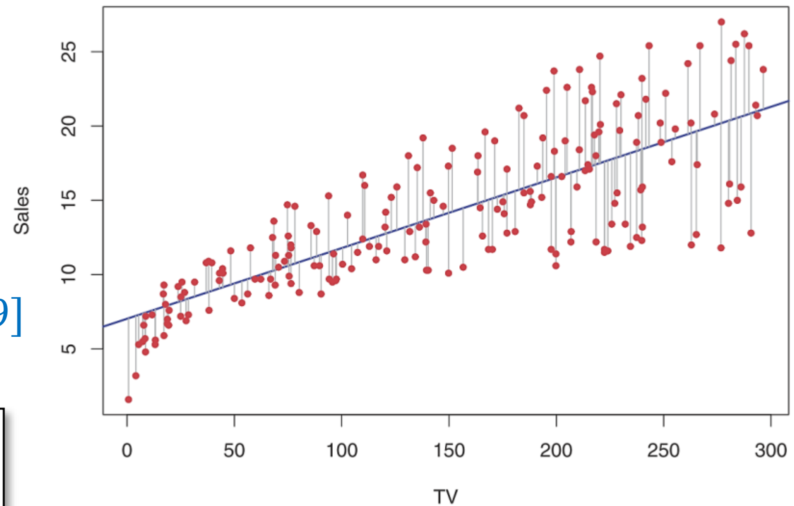
$$\hat{\beta}_0 = 7.0326 \qquad \hat{\beta}_1 = 0.0475$$

while the 95% confidence intervals and the R^2 statistic are:

$$SE(\hat{\beta}_0) = 0.4578 \qquad SE(\hat{\beta}_1) = 0.0027$$

$$CI(\hat{\beta}_0) = [6.1169, 7.9483] \qquad CI(\hat{\beta}_1) = [0.0422, 0.0529]$$

$$R^2 = 0.6118$$




```
import numpy as np
from math import sqrt
import pandas as pd
# www.kaggle.com/datasets/ashydv/advertising-dataset
df = pd.read_csv('datasets/advertising.csv')
print(df.info())
x = df['TV'].to_numpy()
y = df['sales'].to_numpy()
sx = x.sum()
sy = y.sum()
sxy = np.sum(np.multiply(x, y))
sx2 = np.sum(x**2)
N = x.shape[0]
b0 = (sy*sx2 - sx*sxy) / (N*sx2 - sx**2)
b1 = (N*sxy - sx*sy) / (N*sx2 - sx**2)
```

```
yp = b0 * np.ones((N,)) + b1 * x
rss = np.sum((y - yp)**2)
sm2 = rss / (N-2)
xb = sx / N
xvar = np.sum( (x - xb * np.ones((N,))) ** 2 )
seb0 = sqrt(sm2 * sx2 / (N * xvar))
seb1 = sqrt(sm2 / xvar)
cib0 = [b0 - 2*seb0, b0 + 2*seb0]
cib1 = [b1 - 2*seb1, b1 + 2*seb1]
tss = N * np.var(y)
R2 = 1 - rss / tss
```

- With more than one feature (**multiple linear regression**):

$$f(x;\theta=\{\beta_j\}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_L x_L$$
$$= \beta_0 + \sum_{j=1}^{L} \beta_j x_j$$

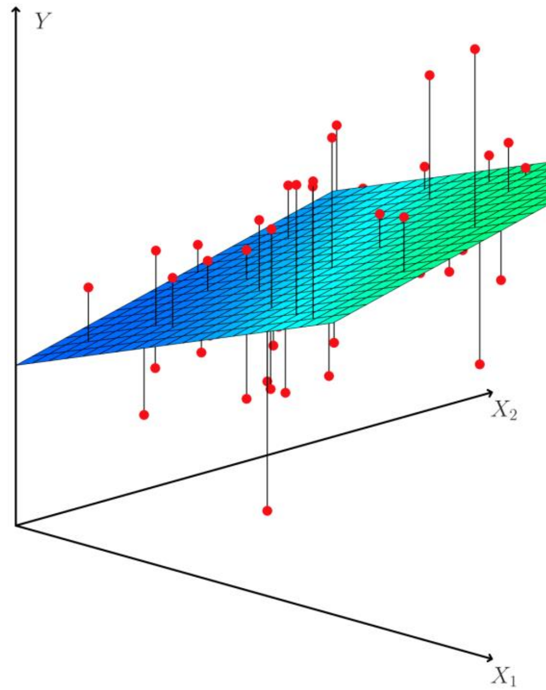
- To solve for θ in

$$\widehat{\theta} = \{\widehat{\beta}_j\} = \arg\min_{\theta}\ L(\theta) = \frac{1}{2}\sum_{i=1}^{N}\left(y_i - (\beta_0 + \sum_{j=1}^{L}\beta_j x_{ij})\right)^2$$

we can adopt matrix notation:

$$\widehat{\theta} = \arg\min_{\theta}\ L(\theta) = \frac{1}{2}\|y - X\theta\|^2 = \frac{1}{2}(y - X\theta)^T (y - X\theta)$$

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_N \end{bmatrix}, X = \begin{bmatrix} 1 & x_{11} & \dots & x_{1j} & \dots & x_{1L} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & x_{i1} & \dots & x_{ij} & \dots & x_{iL} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & \dots & x_{Nj} & \dots & x_{NL} \end{bmatrix}, \theta = \begin{bmatrix} \beta_0 \\ \vdots \\ \beta_j \\ \vdots \\ \beta_L \end{bmatrix}$$



- Taking derivatives and setting them equal to zero as before:

$$\frac{\partial L}{\partial \theta} = -X^T(y - X\theta) = 0 \Rightarrow -X^T y + (X^T X)\theta = 0 \Rightarrow \boxed{\theta = (X^T X)^{-1} X^T y}$$

(since $\partial^2 L / \partial \theta^2 = X^T X$ is positive definite, it is ensured that the optimum is a minimum)

- Matrix $X^+ = (X^T X)^{-1} X^T$ is named as the (Moore-Penrose) **pseudo-inverse** of matrix X

$$\boxed{\theta = X^+ y}$$

- **Example** Let us consider the full advertising dataset, which contains budget data for TV, radio and newspapers advertising, and let us find the fitting hyperplane

```
import numpy as np
import pandas as pd
df = pd.read_csv('datasets/advertising.csv')
print(df.info())
X = df[['TV','radio','newspaper']].to_numpy()
y = df['sales'].to_numpy()
N = X.shape[0]

X_ = np.hstack((np.ones((N,1)), X))
S = np.matmul(X_.T, X_)
Xp = np.matmul(np.linalg.inv(S), X_.T)
th = np.matmul(Xp, y)

rmse = sqrt(((y - X_ @ th) ** 2).sum() / N)
```

Alternatively:

```
Xp = np.linalg.pinv(X_)
th = np.matmul(Xp, y)
```

$$f(x;\theta) = \beta_0 + \beta_1 \times \text{TV} + \beta_2 \times \text{radio} + \beta_3 \times \text{newspaper}$$

$$\theta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} = \begin{bmatrix} 2.9389 \\ 0.0458 \\ 0.1885 \\ -0.0010 \end{bmatrix}$$

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_i (y_i - \widehat{y}_i)^2} = 1.6686$$

Contents

Polynomial regression

- Special case of multiple linear regression which estimates the relationship as an **n-th degree polynomial**:

  e.g. $f(x;\beta_0,\beta_1,\beta_2) = \beta_0 + \beta_1 x + \beta_2 x^2$

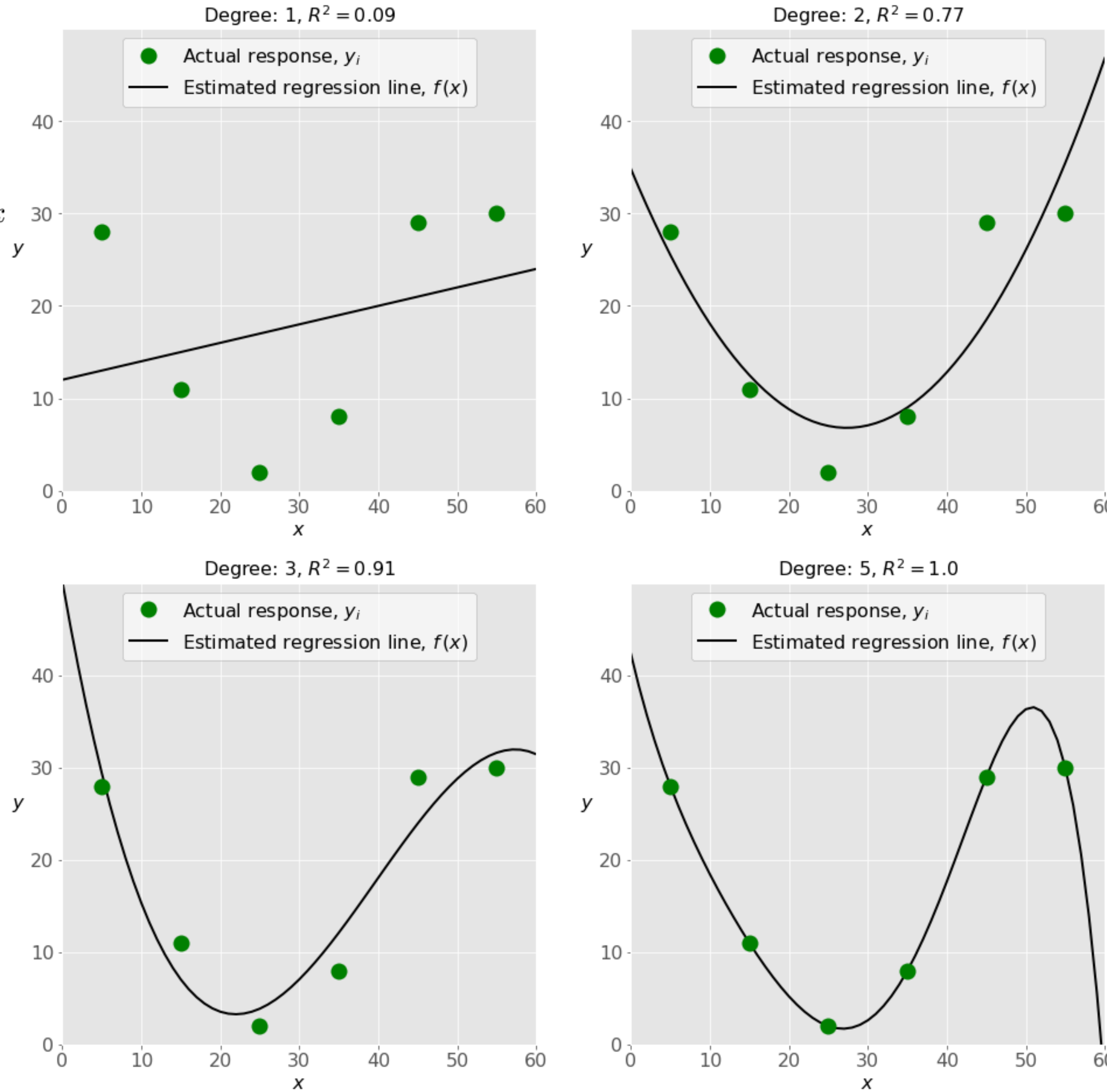
- In general:

  $f(x;\theta=\{\beta_j\}) = \beta_0 + \beta_1 x + \cdots + \beta_k x^k$

- This is also a case of linear regression since we can write:

$$X = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^k \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_i & x_i^2 & \dots & x_i^k \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 & \dots & x_N^k \end{bmatrix}, \theta = \begin{bmatrix} \beta_0 \\ \vdots \\ \beta_j \\ \vdots \\ \beta_k \end{bmatrix}$$

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_N \end{bmatrix} \qquad \boxed{\theta = X^+ y}$$



Polynomial regression

- In the **multivariate/multiple linear regression case**, the number of coefficients grows significantly, but it is the **same kind of optimization problem**

  e.g. for two features x_1 and x_2 and a polynomial of degree 2

$$f(x; \theta = \{\beta_j\}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 x_1 x_2 + \beta_5 x_2^2$$

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & x_{11}^2 & x_{11}x_{12} & x_{12}^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{i1} & x_{i2} & x_{i1}^2 & x_{i1}x_{i2} & x_{i2}^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{N1} & x_{N2} & x_{N1}^2 & x_{N1}x_{N2} & x_{N2}^2 \end{bmatrix}, \theta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \\ \beta_5 \end{bmatrix}$$

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_N \end{bmatrix} \qquad \boxed{\theta = X^+ y}$$

Polynomial regression

- Scikit-learn deals with this latter case **as if it was a first-degree polynomial** (uni- or multi-variate):

```
import pandas as pd
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

df = pd.read_csv("datasets/advertising.csv")
x = df["TV"].to_numpy()
y = df["sales"].to_numpy()
x = x.reshape(-1,1)

trf = PolynomialFeatures(degree=3, include_bias=True)
trf.fit(x)
x_ = trf.transform(x)
lr = LinearRegression()
lr.fit(x_, y)
theta = [lr.intercept_, lr.coef_]

R2 = lr.score(x_, y)
print("R2 = %f" % (R2))
yp = lr.predict(x_)
rmse = mean_squared_error(y, yp, squared=False)
print("RMSE = %f" % (rmse))
```

```
R2   = 0.6220
RMSE = 3.1997
```

ALSO:

```
trf = PolynomialFeatures(degree=3,
                         include_bias=True)
trf.fit(x)
x_ = trf.transform(x)
theta = np.linalg.pinv(x_) @ y
```

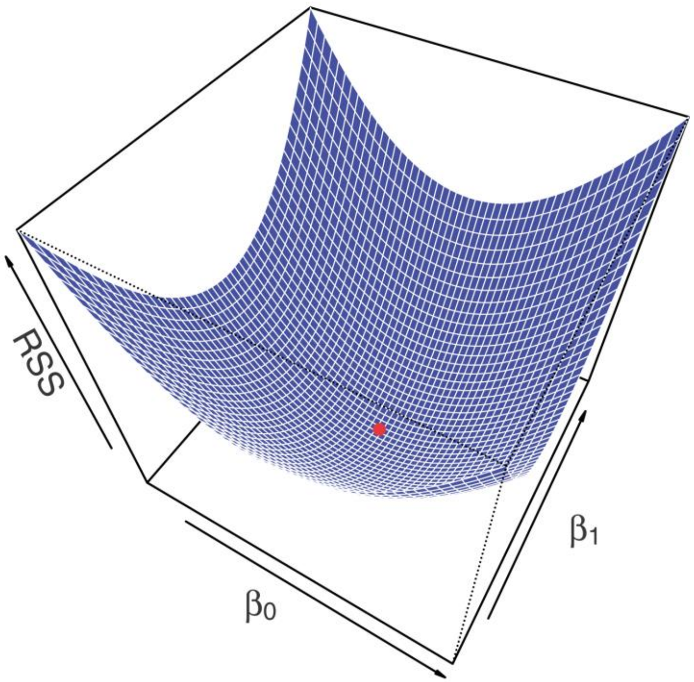
```
X = df[["TV","radio",
        "newspaper"]].to_numpy()
y = df["sales"].to_numpy()

trf = PolynomialFeatures(degree=2,
                         include_bias=False)
trf.fit(X)
X_ = trf.transform(X)
lr = LinearRegression()
lr.fit(X_, y)
```

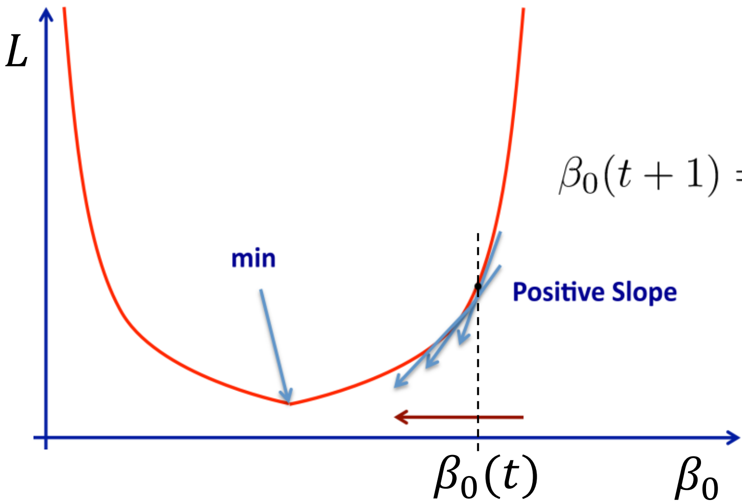
```
R2   = 0.9865
RMSE = 0.6046
```

Contents

- **Gradient descent** is an iterative optimization method that results particularly useful when the optimization problem does not have a closed-form solution (unlike linear regression), e.g. **non-linear optimization**

$$\widehat{\theta} = \{\beta_0, \beta_1\} = \underset{\beta_0,\beta_1}{\arg\min}\ L(\beta_0, \beta_1) = \frac{1}{2}\sum_{i=1}^{N}(y_i - (\beta_0 + \beta_1 x_i))^2$$





$$\beta_0(t+1) = \beta_0(t) - \alpha\frac{\partial L(\beta_0, \beta_1; t)}{\partial \beta_0}$$

- α is named as the **learning rate**
  - Fraction of the derivative that is used to update the parameter
- The same approach can be adopted for β_1:

$$\beta_1(t+1) = \beta_1(t) - \alpha\frac{\partial L(\beta_0, \beta_1; t)}{\partial \beta_1}$$

  - Both β_0 and β_1 are updated simultaneously at every iteration

Gradient descent

- For the **1D linear regression** case (this is not a non-linear optimization case, but we will use it to illustrate the optimization scheme):

$$\widehat{\theta} = \{\beta_0, \beta_1\} = \arg\min_{\beta_0,\beta_1} \; L(\beta_0, \beta_1) = \frac{1}{2}\sum_{i=1}^{N}(y_i - (\beta_0 + \beta_1 x_i))^2$$

$$\left.\begin{aligned} \frac{\partial L}{\partial \beta_0} &= \sum_{i=1}^{N}(y_i - (\beta_0 + \beta_1 x_i))(-1) \\ \beta_0(t+1) &= \beta_0(t) - \alpha \frac{\partial L(\beta_0, \beta_1; t)}{\partial \beta_0} \end{aligned}\right\} \Rightarrow \beta_0(t+1) = \beta_0(t) - \alpha \sum_{i=1}^{N}(\beta_0 + \beta_1 x_i - y_i)$$

$$\left.\begin{aligned} \frac{\partial L}{\partial \beta_1} &= \sum_{i=1}^{N}(y_i - (\beta_0 + \beta_1 x_i))(-x_i) \\ \beta_1(t+1) &= \beta_1(t) - \alpha \frac{\partial L(\beta_0, \beta_1; t)}{\partial \beta_1} \end{aligned}\right\} \Rightarrow \beta_1(t+1) = \beta_1(t) - \alpha \sum_{i=1}^{N}(\beta_0 + \beta_1 x_i - y_i)\, x_i$$

- In the general multi-dimensional case, the set of parameters is updated using the **gradient vector** $\nabla L = (\partial L/\partial \beta_0, \partial L/\partial \beta_1, \ldots, \partial L/\partial \beta_L)$:

$$\boxed{\theta(t+1) = \theta(t) - \alpha \, \nabla L(\theta; t)}$$

- Hence, the **full optimization scheme** consists in:

$$\theta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} = \begin{bmatrix} 3.5110 \\ 0.0430 \\ 0.1733 \\ 0.0072 \end{bmatrix}$$

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_i (y_i - \widehat{y}_i)^2} = 1.7025$$

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

df = pd.read_csv('datasets/advertising.csv')
print(df.info())
X = df[['TV','radio','newspaper']].to_numpy()
y = df['sales'].to_numpy()
N = X.shape[0]

sc = MinMaxScaler()
Xhat = sc.fit_transform(X)

th = gdlinreg(Xhat, y, 0.0001, 1e-3, 2000)

th[0] -= th[1]*sc.min_[0]*sc.scale_[0] \
       - th[2]*sc.min_[1]*sc.scale_[1] \
       - th[3]*sc.min_[2]*sc.scale_[2]
th[1] *= sc.scale_[0]
th[2] *= sc.scale_[1]
th[3] *= sc.scale_[2]

X_ = np.hstack((np.ones((N,1)), X))
rmse = sqrt(((y - X_ @ th) ** 2).sum() / N)
print('RMSE = %f' % (rmse))
```

```
def gdlinreg(X, y, alpha, tol, tmax):
    N = X.shape[0]
    th,th0 = np.zeros(4),np.zeros(4)
    for _ in range(tmax):
        sgrad,grad = np.zeros(4),np.zeros(4)
        for i in range(N):
            x = X[i,:]
            yhat = th[0] + th[1]*x[0] + \
                       th[2]*x[1] + th[3]*x[2]
            e = (yhat - y[i])
            grad[0] = e
            grad[1] = e * x[0]
            grad[2] = e * x[1]
            grad[3] = e * x[2]
            sgrad += grad
        th = th - alpha * sgrad
        if np.max(np.abs(th - th0)) < tol:
            break
        else:
            th0 = th.copy()
    return th
```

- Using **scikit-learn**:

```
(continued)

from sklearn.linear_model import SGDRegressor

sgd = SGDRegressor(loss='squared_error')
sgd.fit(Xhat, y)

th = np.zeros(4)
th[0] = sgd.intercept_ - sgd.coef_[0]*sc.min_[0]*sc.scale_[0] \
                       - sgd.coef_[1]*sc.min_[1]*sc.scale_[1] \
                       - sgd.coef_[2]*sc.min_[2]*sc.scale_[2]
th[1] = sgd.coef_[0]*sc.scale_[0]
th[2] = sgd.coef_[1]*sc.scale_[1]
th[3] = sgd.coef_[2]*sc.scale_[2]

X_ = np.hstack((np.ones((N,1)), X))
rmse = sqrt(((y - X_ @ th) ** 2).sum() / N)
print('RMSE = %f' % (rmse))
```

$$\theta = \begin{bmatrix}\beta_0\\\beta_1\\\beta_2\\\beta_3\end{bmatrix} = \begin{bmatrix}3.5308\\0.0429\\0.1728\\0.0074\end{bmatrix}$$

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_i (y_i - \widehat{y}_i)^2} = 1.7048$$

- **Pros and Cons** of both optimization approaches:

<u>Analytical approach (normal equation)</u>
(+) No need to specify a covergence rate or iterate
(-) Requires a loss function whose derivative exists and leads to a closed-form solvable system of equations
(-) Works only if the pseudo-inverse can be calculated (i.e. $X^T X$ is invertible)

<u>Iterative approach (gradient descent)</u>
(+) Generic approach for every loss function provided the derivative exists
(+) Effective in high dimensions provided there is enough gradient for the updates to happen, i.e. learning to occur (= problems with plateaus)
(-) May require many iterations to converge
(-) Requires to set up $\theta(0)$ before starting the iterative process
(-) Problems with local minima
(-) Requires to set up the learning rate α, do not use a learning rate that is too small or too large

Contents

- This is a classification method that adopts a regression approach to fit a **classification function**. We will consider a **two-class problem**:

$$f : \mathbb{R}^L \longrightarrow \mathbb{Y} = \{0, 1\}$$
$$f(x) = y$$

  – The output is real (regression), but is bounded (classification)

- **Can we use linear regression**?

$$f : \mathbb{R}^L \longrightarrow \mathbb{R}$$
$$f(x) = y$$

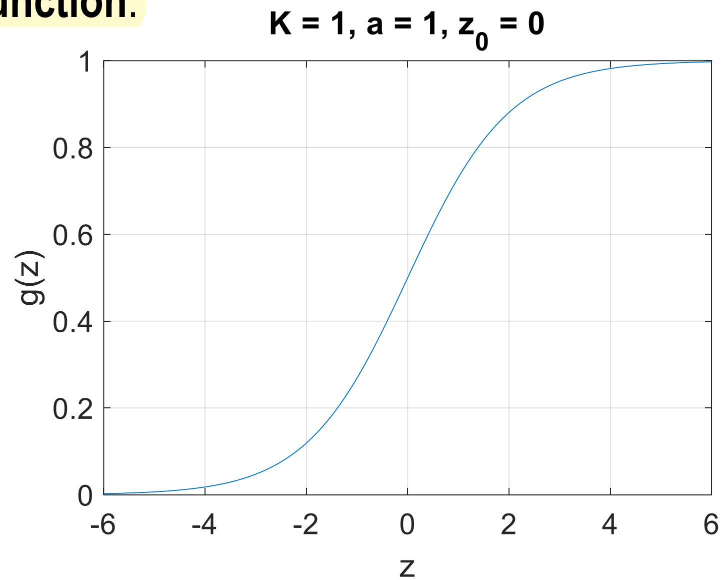
  – Yes, but:

    1. Although $y \in \{0,1\}$, *a priori* $f(x)$ **takes real values not discrete labels**
    2. The linear regression model **does not ensure boundness**, i.e. we need $0 \leq f(x) \leq 1$ but the regressed $f(x)$ can take values out of $[0,1]$

Logistic regression

- It is better to adopt a function that ensure boundness, e.g. a **sigmoid function** (i.e. a S-shaped function) named as the **logistic function**:

$$g(z) = K \frac{e^{a(z-z_0)}}{1+e^{a(z-z_0)}} = \frac{K}{1+e^{-a(z-z_0)}}$$



  - For $K=1$,
    $g(z) \to 1 \; when \; z \to +\infty$ and
    $g(z) \to 0 \; when \; z \to -\infty$

- In the **logistic regression** case:

$$z(x_i;\theta) = \beta_0 + \sum_{j=1}^{L} \beta_j x_{ij}$$

$$g(x_i;\theta) = \frac{1}{1+e^{-z(x_i;\theta)}}$$

For the **1D case**:

$$g(x_i;\beta_0,\beta_1) = \frac{1}{1+e^{-(\beta_0+\beta_1 x_i)}}$$

- Now, we have:

$$g(x_i;\theta) = \frac{1}{1+e^{-z(x_i;\theta)}}\,, \text{ with } z(x_i;\theta) = \beta_0 + \sum_{j=1}^{L} \beta_j x_{ij}$$

$$L(\theta) = \frac{1}{2}\sum_{i=1}^{N}(g(x_i;\theta)-y_i)^2 = \frac{1}{2}\sum_{i=1}^{N}\left(\frac{1}{1+e^{-(\beta_0+\sum_j \beta_j x_{ij})}} - y_i\right)^2$$

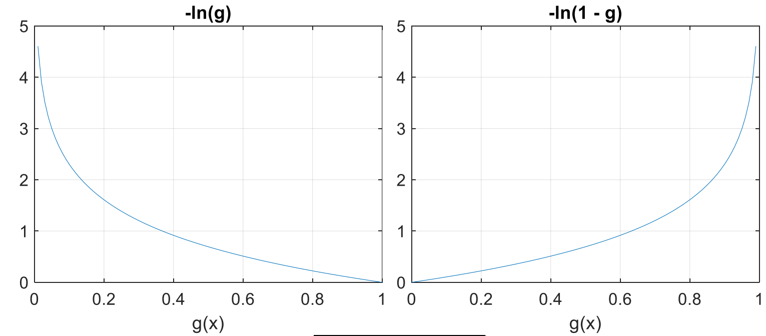
- Because of the presence of $g(x_i)$, the least squares loss $L(\theta)$ is not the same function as for the case of linear regression and hence the least squares loss becomes less appropriate
  - The loss function is now a **more complex non-linear function**
  - There may be **many local optima**, hence gradient descent is not ensured to find the global optimum

- The **binary cross-entropy** loss function is used instead:

$$L_i(\theta) = -\left(y_i \ln g(x_i;\theta) + (1-y_i)\ln(1-g(x_i;\theta))\right)$$

$$L(\theta) = \sum_{i=1}^{N} L_i(\theta)$$

$$= -\sum_{i=1}^{N}\left(y_i \ln g(x_i;\theta) + (1-y_i)\ln(1-g(x_i;\theta))\right)$$



y_i	g_i	
1	1	$L_i = 0$
0	0	$L_i = 0$
1	0	$L_i \to \infty$
0	1	$L_i \to \infty$

- To learn the set of parameters θ, we can adopt gradient descent, which results in a quite **convenient optimization scheme**:

$$L_i = -\left(y_i \ln g(x_i;\theta) + (1-y_i)\ln(1-g(x_i;\theta))\right), \text{ with } g(x_i;\theta) = \frac{1}{1+e^{-z(x_i;\theta)}} \text{ and } z(x_i;\theta) = \beta_0 + \sum_{j=1}^{L} \beta_j x_{ij}$$

$$\frac{\partial L_i}{\partial \theta} = -\left(y_i \frac{1}{g_i}\frac{\partial g}{\partial z}\frac{\partial z}{\partial \theta} + (1-y_i)\frac{(-1)}{1-g_i}\frac{\partial g}{\partial z}\frac{\partial z}{\partial \theta}\right) \qquad \Rightarrow \frac{\partial L_i}{\partial \theta} = -\left(y_i(1-g_i) - (1-y_i)g_i\right)\frac{\partial z}{\partial \theta}$$

$$\frac{\partial g}{\partial z} = \frac{(-1)}{(1+e^{-z})^2}e^{-z}(-1) = \frac{e^{-z}}{(1+e^{-z})^2} = g\,(1-g) \qquad = (g_i - y_i)\frac{\partial z}{\partial \theta}$$

$$\beta_0(t+1) = \beta_0(t) - \alpha\frac{\partial L(\theta;t)}{\partial \beta_0} = \beta_0(t) - \alpha\sum_{i=1}^{N}\left(g(x_i;\theta,t) - y_i\right)\frac{\partial z}{\partial \beta_0} = \beta_0(t) - \alpha\sum_{i=1}^{N}\left(g(x_i;\theta,t) - y_i\right)$$

$$[j>0]\ \beta_j(t+1) = \beta_j(t) - \alpha\frac{\partial L(\theta;t)}{\partial \beta_j} = \beta_j(t) - \alpha\sum_{i=1}^{N}\left(g(x_i;\theta,t) - y_i\right)\frac{\partial z}{\partial \beta_j} = \beta_j(t) - \alpha\sum_{i=1}^{N}\left(g(x_i;\theta,t) - y_i\right)x_{ij}$$

$$\boxed{\begin{aligned} \beta_0(t+1) &= \beta_0(t) - \alpha\sum_{i=1}^{N}\left(g(x_i;\theta,t) - y_i\right) \\ \beta_j(t+1) &= \beta_j(t) - \alpha\sum_{i=1}^{N}\left(g(x_i;\theta,t) - y_i\right)x_{ij} \end{aligned}} \quad [j>0]$$

- Multi-class logistic regression adopts the (categorical) **cross-entropy loss**. For **M** classes:

$$L_i(\theta) = -\sum_{k=1}^{M} t_{ik} \ln g_k(x_i)$$

$$L(\theta) = \sum_{i=1}^{N} L_i(\theta) = -\sum_{i=1}^{N}\sum_{k=1}^{M} t_{ik} \ln g_k(x_i)$$

  where t_i is the **one-hot encoding** for sample x_i.

- The **gradient descent scheme** requires the regression of g_k functions, $k = 1, \ldots, M$

$$g_k(x_i) = g(x_i;\theta_k) = \frac{1}{1+e^{-z(x_i;\theta_k)}}, \text{ with } z(x_i;\theta_k) = \beta_{k,0} + \sum_{j=1}^{L} \beta_{k,j} x_{ij}$$

$$\boxed{\begin{aligned} \beta_0(t+1) &= \beta_0(t) - \alpha \sum_{i=1}^{N}\sum_{k=1}^{M} \left(g_k(x_i;t) - t_{ik}\right) \\ \beta_j(t+1) &= \beta_j(t) - \alpha \sum_{i=1}^{N}\sum_{k=1}^{M} \left(g_k(x_i;t) - t_{ik}\right) x_{ij} \end{aligned}} \quad [j > 0]$$

Logistic regression

- **Example**
  ("manual" gradient descent)

```
import numpy as np
from math import exp
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import OneHotEncoder

X, y = load_iris(return_X_y=True)
y_ = np.expand_dims(y, axis=-1)
ohe = OneHotEncoder()
ohe.fit(y_)
yenc = ohe.transform(y_).toarray()

th = gdlogreg(X, yenc, 0.0001, 1e-4, 3000)
ypred = logregpred(X, th)

print(accuracy_score(y, ypred))
```

```
0.9733
```

class	β_0	β_1	β_2	β_3	β_4
1	+0.2795	+0.4382	+1.4916	−2.3406	−1.0570
2	+0.3623	+0.3701	−1.2753	+0.4169	−0.8631
3	−0.7561	−1.4982	−1.3545	+2.2299	+1.9007

```
def gdlogreg(X, t, alpha, tol, tmax):
  N, L, M = X.shape[0], X.shape[1], t.shape[1]
  th,th0 = np.zeros((L+1,M)),np.zeros((L+1,M))
  for _ in range(tmax):
    sgrad = np.zeros((L+1,M))
    grad = np.zeros((L+1,M))
    for i in range(N):       # loop over the samples
      x = np.array([1,X[i,0],X[i,1],X[i,2],X[i,3]])
      for c in range(M):     # loop over the classes
        z = np.dot(th[:,c], x)
        that = 1/(1 + exp(-z))
        e = (that - t[i,c])
        grad[:,c] = e * x    # grad. of all coeff.
      sgrad += grad
    th = th - alpha * sgrad
    if np.max(np.abs(th - th0)) < tol:
      break
    else:
      th0 = th.copy()
    return th
```

```
def logregpred(X, th):
  N, M = X.shape[0], th.shape[1]
  ypred = np.zeros(N)
  for i in range(N):
    x = np.array([1,X[i,0],X[i,1],X[i,2],X[i,3]])
    that = np.zeros(M)
    for c in range(M):
      z = np.dot(th[:,c], x)
      that[c] = 1/(1 + exp(-z))
    ypred[i] = np.argmax(that)
  return ypred
```

- **Example** (scikit-learn)

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

X, y = load_iris(return_X_y=True)
clf = LogisticRegression()
clf.fit(X, y)
ypred = clf.predict(X)
print(accuracy_score(y, ypred))
```

```
0.9733
```

$class$	β_0	β_1	β_2	β_3	β_4
1	$+9.8500$	-0.4236	$+0.9674$	-2.5171	-1.0794
2	$+2.2372$	$+0.5345$	-0.3216	-0.2064	-0.9442
3	-12.0872	-0.1108	-0.6457	$+2.7235$	$+2.0236$

Lecture 3.2
Supervised learning: Regression

Universitat de les Illes Balears
Departament de Ciències Matemàtiques i Informàtica

11752 Aprendizaje Automático
11752 Machine Learning
Máster Universitario
en Sistemas Inteligentes

Alberto ORTIZ RODRÍGUEZ