

# Supervised Learning

Carlos Valle

Departamento de Informática  
Universidad Técnica Federico Santa María

*cvalle@inf.utfsm.cl*

October 9, 2015

# Overview

- 1 Learning from examples
- 2 Linear Regression
- 3 Matrix derivatives
- 4 Revisited linear regression

# Motivation

- Suppose we have a dataset giving weight, gender and calorie consumption a day from 40 people.

Weight (Kg)	Gender (M/F)	calorie cons. (cal)
85	M	2075
58	F	1757
52	M	2783
55	F	3500
$\vdots$	$\vdots$	$\vdots$

- We would like to predict the calorie consumption per day of other people.
- In this case, weight and gender are called **input features**. Each **vector input**  $\mathbf{x}_m$  has these two features.
- A calorie consumption per day  $y_m$  is called **target**.
- A pair  $(\mathbf{x}_m, y_m)$  is called a train example. A **training set** is a group of  $M$  training examples  $S_M = \{(\mathbf{x}_m, y_m)\}, m = 1 \dots M$ .

# Preliminary definitions

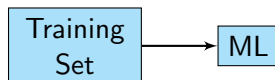
- Features can be either **numerical** or **categorical**.
- Let  $\mathcal{X}$  be the **feature space**, and  $\mathcal{Y}$  the **output space**.
- Our goal is to obtain a mapping  $f : \mathcal{X} \rightarrow \mathcal{Y}$ , commonly called the *hypothesis* or **learner**),  $\mathcal{X} \subseteq \mathbb{R}^n$ ,  $y \in Y \subseteq \mathbb{R}$ .
- Let  $\mathcal{S}$  the space that spans the possible samples, drawn from an unknown distribution  $P(\mathbf{x}, y)$ .
- A *learning algorithm* is a map from the space of training sets to the hypothesis space  $\mathcal{H}$  of possible functional solutions

$$\begin{aligned} \mathcal{A} &: \mathcal{S} \rightarrow \mathcal{H} \\ S_M &\rightarrow \mathcal{A}(S_M) = f. \end{aligned} \tag{1}$$

Training  
Set

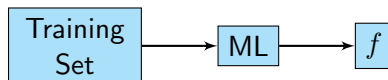
- If the target is continuous, we have a **regression problem**
- If the target can take a finite number  $k$  of discrete values, we have a **classification problem**. In particular  $k = 2$  the problem is called **binary classification**.

# Learning process



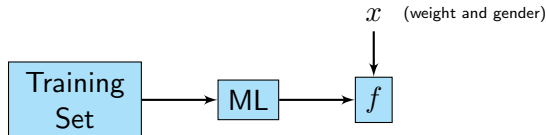
- If the target is continuous, we have a **regression problem**
- If the target can take a finite number  $k$  of discrete values, we have a **classification problem**. In particular  $k = 2$  the problem is called **binary classification**.

# Learning process



- If the target is continuous, we have a **regression problem**
- If the target can take a finite number  $k$  of discrete values, we have a **classification problem**. In particular  $k = 2$  the problem is called **binary classification**.

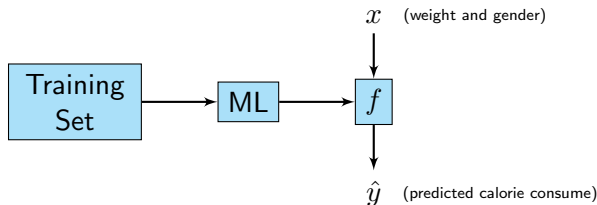
# Learning process



- If the target is continuous, we have a **regression problem**
- If the target can take a finite number  $k$  of discrete values, we have a **classification problem**. In particular  $k = 2$  the problem is called **binary classification**.



# Learning process



- If the target is continuous, we have a **regression problem**
- If the target can take a finite number  $k$  of discrete values, we have a **classification problem**. In particular  $k = 2$  the problem is called **binary classification**.

# Learning process (2)

- A main challenge is to construct an automatic method or **algorithm** able to estimate future examples based on the observed phenomenon in the training set.
- This key property of an algorithm is known as the **generalization ability**.
- The algorithms that memorize the training samples but have poor predictive performance with unknown examples, this undesirable problem is well-known as **overfitting**.

# Loss function

- The quality of the algorithm  $\mathcal{A}$  is measured by the **loss function** given by  $\ell : \mathbb{R} \times \mathcal{Y} \rightarrow [0, \infty)$ , which quantifies the accuracy of the observed response  $f(\mathbf{x})$  with respect to the true or **desired response**  $y$ .
- This function does not penalize the exact predictions, i.e.,  $\ell(y, f(\mathbf{x})) = 0$  if and only if  $y = f(\mathbf{x})$ .
- $\ell$  is a non-negative function, hence, the hypothesis will never profit from additional good predictions.
- In regression settings we use the **quadratic loss** function  $\ell(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$
- While in classification we have the **misclassification loss** function

$$\ell(f(\mathbf{x}), y) = \begin{cases} 0 & \text{if } f(\mathbf{x}) = y \\ 1 & \text{if } f(\mathbf{x}) \neq y \end{cases}.$$

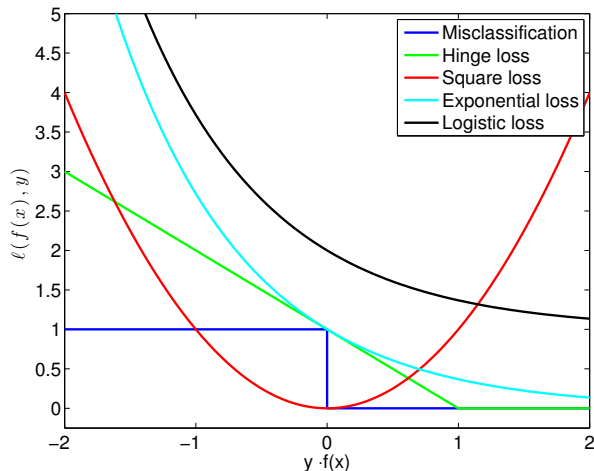
## Loss function (2)

- However, In binary classification problem (where  $y \in \{-1, 1\}$ ), the **margin**  $yf(\mathbf{x})$  is introduced as a quality measure.
- This quality amount leads to several loss functions such as the *hinge loss*

$$\ell(f(\mathbf{x}), y) = \max(1 - yf(\mathbf{x}), 0) = |1 - yf(\mathbf{x})|_+.$$

- The *logistic loss*  $\ell(f(\mathbf{x}), y) = \log_2 (1 + e^{-yf(\mathbf{x})})$
- The *exponential loss*  $\ell(f(\mathbf{x}), y) = e^{-yf(\mathbf{x})}$ .
- Note that the square loss can be arranged as  $\ell(f(\mathbf{x}), y) = (y - f(\mathbf{x}))^2 = (1 - yf(\mathbf{x}))^2$  taking into account that  $y^2 = 1$ .
- And the misclassification loss can be written as  $\ell(f(\mathbf{x}), y) = I(yf(\mathbf{x}) < 0)$ , where  $I(\cdot)$  is the **indicator function**.

# Loss function (3)



- Logistic Loss and exponential loss can be viewed as a continuous approximation of the misclassification function

# Motivation

- We want to predict the total daily travel time of a trucker considering both the distance traveled and the number of deliveries made as input features:

Distance travel time (km)	Number of deliveries	Travel time (hr)
50	4	8.4
75	10	12.3
34	3	6.5
62	5	10.0
$\vdots$	$\vdots$	$\vdots$

- Here the inputs  $\mathbf{x}$  are bi-dimensional vectors. We let  $\mathbf{x}_m^{(i)}$  denote the feature  $i$  of the  $m$ -th example.

- We could approximate the total daily travel time  $y$  as a linear function of the distance traveled and the number of deliveries made:

$$f(x) = \beta_0 + \beta_1 x^{(1)} + \beta_2 x^{(2)},$$

- where  $\beta_i, i = 0, 1, 2$  are the parameters of the linear model.
- Thus, the space of linear functions mapping from  $\mathcal{X}$  to  $\mathcal{Y}$  is **parametric**.
- We define  $x^{(0)} = 1$  to define the linear model in matrix form:

$$f(x) = \sum_{i=0}^I \beta_i x^{(i)} = \beta^T \mathbf{x}, \quad (2)$$

- where  $I$  is the number of features.

# How do we select the $\beta$ 's?

- We can get the parameters of the model by minimizing the quadratic loss function over the training set:

$$J(\beta) = \frac{1}{2} \sum_{m=1}^M (f(\mathbf{x}_m) - y_m)^2 \quad (3)$$

- where  $\beta = (\beta_1, \beta_2, \dots, \beta_I)^T$ .
- We need to choose  $\beta$  which minimizes  $J(\beta)$ .



- This problem can be expressed in matrix form:

$$J(\beta) = (y - X\beta)^T (y - X\beta),$$

- where  $X$  is an  $N \times (I + 1)$  matrix.

## From matrix calculus

- If  $x$  is a column vector:
- $\frac{\partial u^T v}{\partial x} = \frac{\partial u}{\partial x} \cdot v + \frac{\partial v}{\partial x} \cdot u$
- $\frac{\partial Ax}{\partial x} = A^T$

- Then,

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

- Equalizing the first derivative to zero we get the normal equations:

$$X^T(y - X\beta) = 0 \implies \hat{\beta} = (X^T X)^{-1} X^T y$$

# Gradient descent algorithm

- Let consider the **gradient descent** algorithm which start with some initial  $\beta$  and repeatedly performs:

$$\beta_i^{p+1} = \beta_i^p - \alpha \frac{\partial}{\partial \beta_i} J(\beta)$$

- Let's calculate the derivative of the loss function for a single training example  $(\mathbf{x}_m, y_m)$ :

$$\begin{aligned} \frac{\partial}{\partial \beta_i} J(\beta) &= \frac{\partial}{\partial \beta_i} \frac{1}{2} (f(\mathbf{x}_m) - y_m)^2 \\ &= (f(\mathbf{x}_m) - y_m) \cdot \frac{\partial}{\partial \beta_i} \left( \sum_{i=0}^I \beta_i x^{(i)} \right) \\ &= (f(\mathbf{x}_m) - y_m) \mathbf{x}_m^{(i)} \end{aligned}$$

# Gradient descent algorithm (2)

- Then, for a single example:

$$\beta_i^{p+1} = \beta_i^p - \alpha(f(\mathbf{x}_m) - y_m) \mathbf{x}_m^{(i)}$$

- This rule is called Widrow-Hoff learning rule.
- Note that the amount of the update is proportional to the error:  
 $(f(\mathbf{x}_m) - y_m)$

# Batch gradient descent algorithm

- We can apply the learning rule above for the training set:

---

## Algorithm 1 Batch gradient descent algorithm

---

```
1: repeat  
2:    $\beta_i^{p+1} = \beta_i^p - \alpha \sum_{m=1}^M (f(\mathbf{x}_m) - y_m) \mathbf{x}_m^{(i)}$  (for every  $i$ )  
3: until Convergence
```

---

- In general, gradient descent can reach a **local minimum**.
- However,  $J$  is a convex function. Thus, the optimization problem has only one **global optimum**.

# Stochastic gradient descent algorithm

- We can modify the learning rule above for each example:

---

**Algorithm 2** Stochastic gradient descent algorithm

---

```
1: repeat  
2:   for  $m := 1$  to  $M$  do  
3:      $\beta_i^{p+1} := \beta_i^p - \alpha(f(\mathbf{x}_m) - y_m) \mathbf{x}_m^{(i)}$  (for every  $i$ )  
4:      $\beta_i^p := \beta_i^{p+1}$   
5:   end for  
6: until Convergence
```

---

- This method is called **stochastic** or **online** gradient descent.
- Usually, this technique converge faster than batch gradient descent.
- However, using a fixed value for  $\alpha$  it may never converge to the minimum of  $J(\beta)$ , oscillating around it.
- To avoid this behavior, it is recommended to slowly decrease  $\alpha$  to zero along the iterations.

# Motivation (Matrix derivatives)

- When we find the parameter vector that minimize a loss function, we need to take its derivatives respect to this vector and set them to zero.
- In order to do this, we need to learn some tools fro doing matrix calculus.

# Matrix derivatives

- Let  $f : \mathbb{R}^{P \times Q} \mapsto \mathbb{R}$  be a function which maps from  $P \times Q$  matrices to the real numbers.
- We define the derivative of  $f$  with respect to  $A$  as:

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \cdots & \frac{\partial f}{\partial A_{1Q}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial A_{P1}} & \cdots & \frac{\partial f}{\partial A_{PQ}} \end{bmatrix}$$

- Thus, the gradient  $\nabla_A f(A)$  is an  $P \times Q$  matrix where its  $(i, j)$  element is  $\frac{\partial f}{\partial A_{ij}}$

- For example: Let  $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ , and

$$f(x) = 3A_{11}^2 + A_{12}A_{21} + \frac{1}{2}A_{22}$$

- Then the gradient is  $\nabla_A f(A) = \begin{bmatrix} 6A_{11} & A_{21} \\ A_{12} & \frac{1}{2} \end{bmatrix}$



# Matrix derivatives (2)

- Since our loss functions are in  $\mathbb{R}$ , we will use the **trace operator** to compute the derivatives with respect to the parameter vector.
- Now we will define this operator and we will study its properties.
- After that, we will apply trace derivatives in order to minimize  $J(\beta)$ .

- Let  $A$  a  $P \times P$  matrix.
- The trace of  $A$  is an linear  $M(\mathbb{R}^P) \mapsto \mathbb{R}$  operator defined as:

$$\text{tr}(A) = \sum_{p=1}^P A_{pp}$$

- Thus,  $\text{tr}(A + B) = \text{tr}(A) + \text{tr}(B)$
- And  $\text{tr}(cA) = c\text{tr}(A)$ , where  $c \in \mathbb{R}$
- Also note that  $\text{tr}(c) = c$

- Transposition of dependent variables:

$$\text{tr}(\mathbf{A}) = \text{tr}(\mathbf{A}^T). \quad (4)$$

- Hence,

$$\frac{\partial \text{tr}(\mathbf{A})}{\partial \mathbf{X}} = \frac{\partial \text{tr}(\mathbf{A}^T)}{\partial \mathbf{X}}$$

- Cyclic permutation: Let  $\mathbf{A}, \mathbf{B}$  be two matrices such that  $\mathbf{AB}$  is square. Then,  $\text{tr}(\mathbf{AB}) = \text{tr}(\mathbf{BA})$  Thus,

$$\frac{\partial \text{tr}(\mathbf{AB})}{\partial \mathbf{X}} = \frac{\partial \text{tr}(\mathbf{BA})}{\partial \mathbf{X}}$$

- Moreover,  $\text{tr}(\mathbf{ABC}) = \text{tr}(\mathbf{CAB}) = \text{tr}(\mathbf{BCA})$

## Trace properties (2)

- Transposition of independent variables:

$$\nabla_{\mathbf{A}} f(\mathbf{A}) = \frac{\partial f}{\partial A_{pq}}, p = 1, \dots, P, q = 1, \dots, Q.$$

- Thus,

$$\nabla_{\mathbf{A}^T} f(\mathbf{A}) = \frac{\partial f}{\partial A_{qp}}, p = 1, \dots, P, q = 1, \dots, Q. = (\nabla_{\mathbf{A}} f(\mathbf{A}))^T \quad (5)$$

- Property 2:

$$\nabla_{\mathbf{B}} \text{tr}(\mathbf{A}\mathbf{B}) = \mathbf{A}^T \quad (6)$$

- This came from the fact that

$$D_{\mathbf{Y}} f(\mathbf{X}) = \lim_{t \rightarrow 0} \frac{f(\mathbf{X} + t\mathbf{Y}) - f(\mathbf{X})}{t} = \text{tr}(\mathbf{Y}^T \mathbf{U})$$

- where  $\mathbf{U} = \nabla_{\mathbf{X}} f(\mathbf{X})$

# Trace properties (3)

- Setting  $f(\mathbf{B}) = \text{tr}(\mathbf{AB})$  we have

$$\begin{aligned} D_{\mathbf{Y}}\text{tr}(\mathbf{AB}) &= \lim_{t \rightarrow 0} \frac{\text{tr}(\mathbf{A}(\mathbf{B} + t\mathbf{Y})) - \text{tr}(\mathbf{AB})}{t} \\ &= \lim_{t \rightarrow 0} \frac{\text{tr}(\mathbf{AB} + t\mathbf{AY}) - \text{tr}(\mathbf{AB})}{t} \\ &= \lim_{t \rightarrow 0} \frac{\text{tr}(\mathbf{AB}) + t\text{tr}(\mathbf{AY}) - \text{tr}(\mathbf{AB})}{t} \\ &= \lim_{t \rightarrow 0} \frac{t\text{tr}(\mathbf{AY})}{t} \\ &= \text{tr}(\mathbf{AY}) \\ &= \text{tr}((\mathbf{AY})^T) \\ &= \text{tr}(\mathbf{Y}^T \mathbf{A}^T) \end{aligned}$$

- Thus,  $\nabla_{\mathbf{B}}\text{tr}(\mathbf{AB}) = \mathbf{A}^T$ .

# Product rule

- We can generalize the product rule of scalars to matrices:

$$\frac{\partial UV}{\partial x} = \frac{\partial U}{\partial x} V + U \frac{\partial V}{\partial x}$$

- Besides, from the definition of the gradient we have

$$\text{tr} \left( \frac{\partial Y}{\partial x} \right) = \frac{\partial(\text{tr}(Y))}{\partial x}$$

- Then,

$$\frac{\partial \text{tr}(UV)}{\partial x_{ij}} = \frac{\partial \text{tr}(UV_c)}{\partial x_{ij}} + \frac{\partial \text{tr}(U_c V)}{\partial x_{ij}}$$

- Where the subscript "c" means constant for purposes of differentiation.
- From the last two equations we obtain

$$\frac{\partial \text{tr}(UV)}{\partial X} = \frac{\partial \text{tr}(UV_c)}{\partial X} + \frac{\partial \text{tr}(U_c V)}{\partial X}$$

## Product rule (2)

•

$$\begin{aligned}\nabla_A(\text{tr}(ABA^TC)) &= \nabla_A(\text{tr}(A(BA^TC)_c)) + \nabla_A(\text{tr}(A_cBA^TC)) \\ &= \nabla_A(\text{tr}((BA^TC)_c)A) + \nabla_A(\text{tr}(A^TCAB)) \\ &= C^TAB^T + CAB\end{aligned}$$

•

$$\nabla_A(\text{tr}(|A|)) = |A|(A^{-1})^T \quad (7)$$

- For a detailed proof of the properties above refer to “With(out) A Trace Matrix Derivatives the Easy Way”. Steven W. Nydick.

# Revisited linear regression

- Let's compute the derivative of  $J(\beta)$  with respect to  $\beta$  by using trace derivatives properties:

$$\begin{aligned}\nabla_{\beta} J(\beta) &= \frac{1}{2} \nabla_{\beta} (X\beta - Y)^T (X\beta - Y) \\ &= \frac{1}{2} \nabla_{\beta} (\beta^T X^T X \beta - \beta^T X^T Y - Y^T X \beta + Y^T Y)\end{aligned}$$

- Note that  $\nabla_{\beta} J(\beta)$  is a real number. Then,

$$\nabla_{\beta} J(\beta) = \frac{1}{2} \text{tr}(\nabla_{\beta} (\beta^T X^T X \beta - \beta^T X^T Y - Y^T X \beta + Y^T Y)).$$

From transposition of dependent variables property,

$$\nabla_{\beta} J(\beta) = \frac{1}{2} (\nabla_{\beta} (\text{tr} \beta^T X^T X \beta - 2\text{tr}(Y^T X \beta))).$$



## Revisited linear regression (2)

- Using properties 2 (with  $A^T = \beta$ ,  $B = X^T X$  and  $C = I$ ), and 3 we obtain

$$\begin{aligned}\nabla_{\beta} J(\beta) &= \frac{1}{2} (X^T X \beta + X^T X \beta - 2X^T Y) \\ &= X^T X \beta - X^T Y\end{aligned}$$

- Setting this to zero,

$$\beta = (X^T X)^{-1} X^T Y$$

# Probabilistic linear model

- The use of equation (2) to model the regression problem is not flexible (we can't model underlying errors)
- Thus, we can fix it by introducing random variables:

$$y_m = \sum_{i=0}^I \beta_i x_m^{(i)} + \epsilon_m = \beta^T x_m + \epsilon_m$$

- Where  $\epsilon_m$  represents either unmodeled effects or noise
- Using this new model we could assume that  $\beta$  is a random vector. Hence, we could measure the bias and the variance of  $\beta$  estimators. (We will do that later)

# Probabilistic linear model (2)

- Now, let's assume that the  $\epsilon_m$  are IID (independently and identically distributed) and  $\epsilon_m \sim N(0, \sigma^2)$
- Then, the pdf of  $\epsilon_m$  is given by

$$p(\epsilon_m) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\epsilon_m^2}{2\sigma^2}}$$

- Thus,

$$p(y_m|x_m; \beta) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_m - \beta^T x_m)^2}{2\sigma^2}}$$

# Maximum likelihood

- Given the input (design) matrix  $X$  and the target vector  $Y$ . How we can choose  $\beta$ ?
- It is reasonable to pick the  $\beta$  which maximizes the **likelihood function**:
- 

$$\begin{aligned} L(\beta) &= \prod_{m=1}^M p(y_m|x_m; \beta) \\ &= \prod_{m=1}^M \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_m - \beta^T x_m)^2}{2\sigma^2}} \end{aligned}$$

- As usual we will maximize the **log likelihood**  $\ell(\beta)$  instead:

# Maximum log likelihood

- As usual we will maximize the **log likelihood**  $\ell(\beta)$  instead:



$$\begin{aligned}\ell(\beta) &= \sum_{m=1}^M \log \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_m - \beta^T x_m)^2}{2\sigma^2}} \\ &= M \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{m=1}^M (y_m - \beta^T x_m)^2\end{aligned}$$

- Hence, we obtain the same solution as minimizing (3).

# Locally weighted linear regression

- Instead of minimizing  $J(\beta)$ , we want to minimize

$$= \frac{1}{2} \sum_{m=1}^M w_m (f(\mathbf{x}_m) - y_m)^2 \quad (8)$$

- A standard choice of  $w_m$  is:

$$w_m = e^{(-(x_m - x)^T (x_m - x) / (2\tau^2))},$$

- where  $\tau$  is called **bandwidth**.
- Note that the weights depend on the particular point  $x$  that we are trying to evaluate.
- The solution is computed as  $\beta = (X^T W X)^{-1} X^T W Y$
- This is a non-parametric model.

# Any questions?

