

# How to do it ?: Linear Models

Nicolás Pérez de la Blanca  
DECSAI

# Linear Models: General setting

- $\mathcal{H}$  is the class of all hyperplanes (linear functions of the predictors).
- This is,  $h(\mathbf{x}) = w_0 + w_1x_1 + \cdots + w_dx_d = w_0 + \sum_{i=1}^d x_iw_i$ 
  - Equivalently using  $\mathbf{x}^T = (1, x_1, x_2, \dots, x_d)$ ,  $\mathbf{w}^T = (w_0, w_1, w_2, \dots, w_d)$
  - We can write:  $h(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$
- Three different problems can be approached using this class of functions:
  - **Regression**
  - **Classification,**
  - **Probability estimation.**
- A rule of thumb, when faced with learning problems: it is generally a **winning strategy to try a linear model first.**

# Regression

# Credit -Again

**Classification:** Credit approval (yes/no)

**Regression:** Credit line (dollar amount)

Input:  $\mathbf{x} =$

age	23 years
annual salary	\$30,000
years in residence	1 year
years in job	1 year
current debt	\$15,000
...	...

Linear regression output:  $h(\mathbf{x}) = \sum_{i=0}^d w_i x_i = \mathbf{w}^\top \mathbf{x}$

# Linear Regression

- Now  $\mathcal{X} = \{1\} \times \mathbb{R}^d$ ,  $\mathcal{Y} = \mathbb{R}$  and  $h: \mathcal{X} \rightarrow \mathcal{Y}$

$$E_{out}(h) = \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{P}}[(h(\mathbf{x}) - y)^2]$$



- The  $\mathcal{H}$  class takes the form:  $h(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$

- We want find  $\hat{h}$  such that,  $\hat{h} = \min_{h \in \mathcal{H}} E_{out}(h)$

- In order to do this we minimize the Empirical Risk (ERM):

$$E_{in}(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^N (\mathbf{w}^T \mathbf{x}_i - y_i)^2$$

$$\mathbf{w}_{lin} = \min_{\mathbf{w} \in \mathbb{R}^{d+1}} E_{in}(\mathbf{w})$$

Relevant question: Why does the minimum of  $E_{in}$  guarantee a minimum in  $E_{out}$  ?

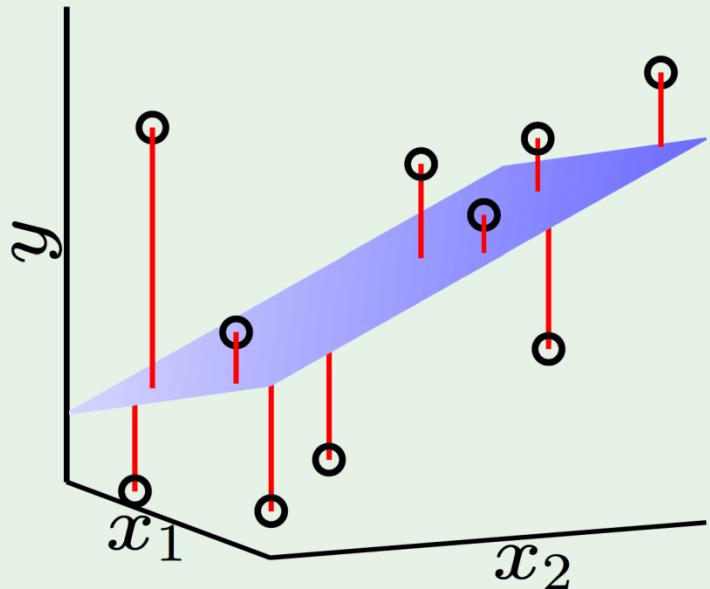
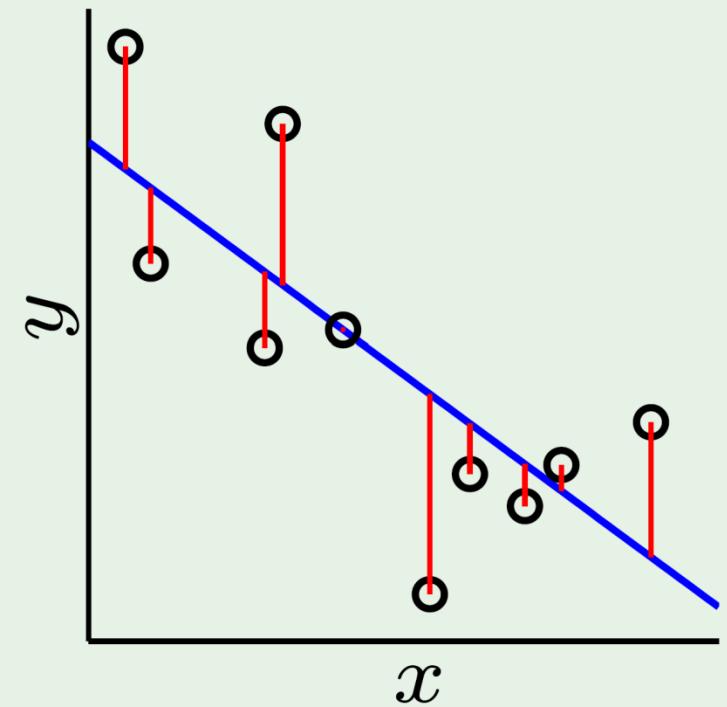
# Ordinary Least Square (OLS): properties

- The OLS model only assumes error in the dependent variable
  - This hypothesis is not valid in all cases
  - But is a good approximation in many cases
- Model:  $y_i = f(\mathbf{x}_i, \boldsymbol{\beta}) + \text{noise}$ ,  $f$  linear in  $\boldsymbol{\beta}$
- **Theorem of Gauss–Markov:** Under the assumption of **in correlated noise, mean zero and bound variance**, the OLS technique reaches the minimum variance unbiased estimator for  $\boldsymbol{\beta}$ 

*Equations:*

  - $y_i = \mathbf{x}_i^T \boldsymbol{\beta} + \epsilon_i$ ,
  - $E(\epsilon_i) = 0$ ,  $E(\epsilon_i \epsilon_j) = 0$ ,  $Var(\epsilon_i^2) = \sigma^2 < \infty$
  - $\hat{\boldsymbol{\beta}}$  - *OLS gives the minimum variance estimator such that  $E(\hat{\beta}_i) = \beta_i$  (unbiased)*

# Illustration of Linear Regression



# A matrix expression for $E_{\text{in}}$

$$\begin{aligned} E_{\text{in}}(\mathbf{w}) &= \frac{1}{N} \sum_{n=1}^N (\mathbf{w}^\top \mathbf{x}_n - y_n)^2 \\ &= \frac{1}{N} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 \end{aligned}$$

where

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_N^\top \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

# Notation

- $\|A\| = \sqrt{\sum_{ij} a_{ij}^2}$
- $\sum_{ij} (x_{ij}w_j - y_i)^2 = \sum_i (x_i^T w - y_i)^2 = \|Xw - y\|^2$
- $\|Xw - y\|^2 = (Xw - y)^T (Xw - y)$ 
$$= (Xw)^T Xw - y^T Xw - (Xw)^T y + y^T y$$
$$= w^T X^T Xw - 2y^T Xw + y^T y$$

# Parameter estimation: Gradient Descent

- We want to choose  $\mathbf{w}$  so as to minimize  $E_{in}(\mathbf{w})$
- Gradient Descent (GD):
  - Gradient descent is a general iterative optimization technique that reaches a local optimum following the direction of the gradient vector on each point.

It starts on some initial value  $\mathbf{w}$  and repeatedly perform the update ,

$$w_j := w_j - \eta \frac{\partial E_{in}(\mathbf{w})}{\partial w_j} \quad (\text{GENERAL EQUATION})$$

(This update is simultaneously performed for all values of  $j = 0, \dots, n$ ). Here,  $\eta$  is called the learning rate.

$$\frac{\partial E_{in}(\mathbf{w})}{\partial w_j} = \frac{\partial}{\partial w_j} \frac{1}{N} \sum_{n=1}^N (\mathbf{w}^T \mathbf{x}_n - y_n)^2 = \frac{2}{N} \sum_{n=1}^N x_{nj} (\mathbf{w}^T \mathbf{x}_n - y_n) = \frac{2}{N} \sum_{n=1}^N x_{nj} (\mathbf{h}(\mathbf{x}_n) - y_n)$$

Each point  $(\mathbf{x}_n, y_n)$  contributes to the update by an amount proportional to its prediction error

In this case all points are used to compute the gradient: **BATCH GRADIENT DESCENT**

# Stochastic Gradient Descent: SGD

- An alternative is to use a **stochastic estimation** using only a part of the sample to compute the gradient,  $M \ll N$  (**SGD**)

$$\frac{\partial E_{in}(\mathbf{w})}{\partial w_j} = \frac{2}{M} \sum_{n=1}^M x_{nj} (\mathbf{h}(\mathbf{x}_n) - y_n)$$

- Higher variability in the gradient estimation ( less examples in the average)
- Very fast of computing
- In non-convex funtions empirical evidence of getting good local minimum
- Although an only item could be used on each iteration, a minibatch of items is the accepted rule ( size: 32-128)

# An iterative regression algorithm

## Batch Gradient Descend

- Given the data set  $(\mathbf{x}_n, y_n), n = 1, 2, \dots, N$
- Fix  $\mathbf{w}=0$ ,  $\eta = \eta_0$
  - Iterate
    - For  $j=0,..,K$  :  
$$w_j := w_j - \eta \sum_{n=1}^N x_{nj} (\mathbf{h}(\mathbf{x}_n) - y_n)$$
  - Until  $E_{in}(\mathbf{w}) < \text{epsilon}$

## Stochastic Gradient Descend

- Fix  $\mathbf{w}=0$ ,  $\eta = \eta_0$
- Fix a sequence of minibatches
- Iterate on minibatches
  - For  $j=0,..,K$  :  
$$w_j := w_j - \eta \sum_{n \in \text{Minibatch}} x_{nj} (\mathbf{h}(\mathbf{x}_n) - y_n)$$
- Until  $E_{in}(\mathbf{w}) < \text{epsilon}$

# Newton's Method

- A new update rule for  $\mathbf{w}$  based on the **second order derivatives** (Hessian)
- The function  $E_{\text{in}}(\mathbf{w})$  is approached on each point by a second order function and we move to the minimum of the function.

$$g(\Delta\mathbf{w}) = E_{\text{in}}(\mathbf{w}_0 + \Delta\mathbf{w}) = E_{\text{in}}(\mathbf{w}_0) + \Delta\mathbf{w}^T \nabla E_{\text{in}}(\mathbf{w}_0) + \frac{1}{2} \Delta\mathbf{w}^T \nabla^2 E_{\text{in}}(\mathbf{w}_0) \Delta\mathbf{w}$$

$$\frac{\partial g(\Delta\mathbf{w})}{\partial \Delta\mathbf{w}} = \mathbf{0} \rightarrow \nabla E_{\text{in}}(\mathbf{w}_0) + \underbrace{\nabla^2 E_{\text{in}}(\mathbf{w}_0)}_{\mathbf{H}} \Delta\mathbf{w} = \mathbf{0}$$

$$\Delta\mathbf{w} = -\mathbf{H}^{-1} \nabla E_{\text{in}}(\mathbf{w}_0)$$

Now the matrix  $\mathbf{H}$  defines the advance on the gradient direction

The major complexity is to invert  $\mathbf{H}$

# Minimizing $E_{\text{in}}$ : normal equations

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$$

$$\nabla E_{\text{in}}(\mathbf{w}) = \frac{2}{N} \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) = 0$$

$$\mathbf{X}^\top \mathbf{X}\mathbf{w} = \mathbf{X}^\top \mathbf{y}$$

$$\mathbf{w} = \mathbf{X}^\dagger \mathbf{y} \quad \text{where} \quad \mathbf{X}^\dagger = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top$$

$\mathbf{X}^\dagger$  is the '**pseudo-inverse**' of  $\mathbf{X}$

# Math details

- To compute  $\frac{\partial E_{in}}{\partial \mathbf{w}}$  we make use of two well known properties:
  - $\frac{\partial}{\partial \mathbf{x}} (\mathbf{x}^T A \mathbf{x}) = (A^T + A)\mathbf{x}, \quad \frac{\partial}{\partial \mathbf{x}} (\mathbf{x}^T A) = A$
- $\frac{\partial E_{in}}{\partial \mathbf{w}} = 2X^T X \mathbf{w} - 2X^T \mathbf{y} = \mathbf{0} \rightarrow \mathbf{w}_{\text{lin}} = (X^T X)^{-1} X^T \mathbf{y}$
- Can we always compute  $(X^T X)^{-1}$ ?
  - Let consider the Singular Value Decomposition (SVD) :  $X=UDV^T$
  - $X^T X = V D D V^T$
  - $(X^T X)^{-1} = V^{-T} D^{-1} V^T = V D^{-1} V^T$  this product can always be computed
  - When the  $\text{range}(X^T X) = d$ , there exist only one solution.
  - When the  $\text{range}(X^T X) < d$ , there exist infinite solutions

# A Linear Regression Algorithm

- 1: Construct the matrix  $\mathbf{X}$  and the vector  $\mathbf{y}$  from the data set  $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)$  as follows

$$\mathbf{X} = \underbrace{\begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_N^\top \end{bmatrix}}_{\text{input data matrix}}, \quad \mathbf{y} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{\text{target vector}}.$$

- 2: Compute the pseudo-inverse  $\mathbf{X}^\dagger = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top$ .
- 3: Return  $\mathbf{w} = \mathbf{X}^\dagger \mathbf{y}$ .

# Linear Regression properties

- An expression in closed-form for all interesting linear regression results can be found ( see exercises in the book)
- **The HAT-MATRIX (  $\hat{H}$  ):** Let's compute the samples prediction. Let  $X (N \times (d+1))$  denote the samples

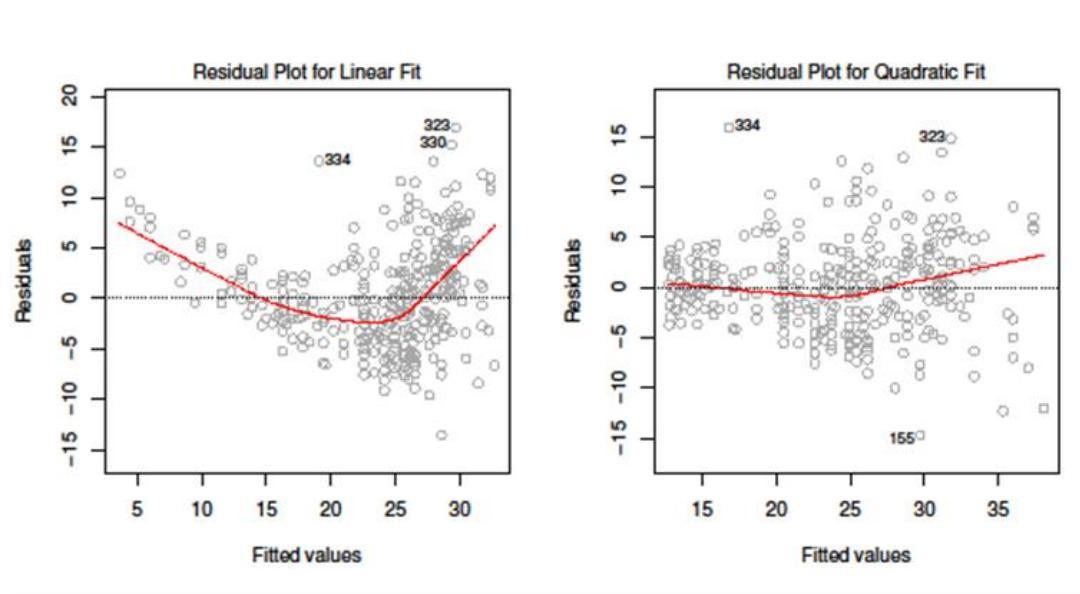
$$\hat{\mathbf{y}} = X\mathbf{w}_{lin} = X(X^T X)^{-1} X^T \mathbf{y} = \hat{H}\mathbf{y}$$

- The matrix  $\hat{H}$  defines, from  $X$ , the weights of the learning labels .
- Linear regression prediction is a weighted linear combination of the training labels.
  - **The hat matrix is an idempotent projection matrix**, this means  $\hat{H}^2 = \hat{H}$
- **Generalization Error:** For linear regression exact formulas for the expected  $E_{out}$  and  $E_{in}$  can be found

$$E_{out}(\mathbf{w}_{lin}) = E_{in}(\mathbf{w}_{lin}) + \mathcal{O}\left(\frac{d}{N}\right)$$

# Assesing the model

- How to assess the quality of the fit ?
- We can analyze the residual, this is the graphical defined by the points  $\{(i, y_i - w_{lin}x_i^T), i = 1, \dots, N\}$



The residues of a good fit show no tendency.

We will study alternative analytic criteria later on

# Polynomial Regression

- In this case the predictors are powers of a single feature, for instance

$$y = \beta_0 + \beta_1 X + \beta_2 X^2 \quad (\text{quadratic model})$$

$$y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 \quad (\text{cubic model})$$

- Clearly all predictors are independent
- The matrix  $X$  for sample data of size  $N$  is defined as

$$X = \begin{bmatrix} 1 & x_1 & \dots & x_1^p \\ 1 & x_2 & \dots & x_2^p \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & \dots & x_N^p \end{bmatrix}$$

- How many terms to choose ? We talk on it later on.

# Fitting a linear regression model

- Let assume a sample  $\mathcal{D}$  of feature vectors from a probability distribution  $\mathbb{P}(\mathcal{X} \times \mathcal{Y})$ . We want to fit a regression model  $\{\mathcal{H}, \mathcal{A}\}$  from  $\mathcal{D}$  to predict  $\mathcal{Y}$  from  $\mathcal{X}$ .

## BASIC STEPS:

- Fix a hypothesis class  $\mathcal{H}$  (with finite VC-dimension)
- Select the potential relevant features in  $\mathcal{X}$  eliminating those features uncorrelated with the  $\mathcal{Y}$  values.
- Fit the model using the retained variables
- Compute the in-sample error
- Analyze the residual errors to determine possible improvements:
  - Go to 6 if no possible improvement is detected
  - Identify new variable transformations to include.
  - Refit the model
  - Go to 4.
- Stop

# Summary of Linear Regression

- Now  $\mathcal{X} = \{1\} \times \mathbb{R}^d$ ,  $y = \mathbb{R}$  and  $h: \mathcal{X} \rightarrow \mathcal{Y}$
- We assume i.i.d. samples from  $\mathcal{X} \times \mathcal{Y}$
- All functions in  $\mathcal{H}$  take the form:  $h(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$
- Learning criteria (ERM):  $\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^N (\mathbf{w}^T \mathbf{x}_i - y_i)^2$
- Quadratic loss/error function:  $\text{error}_h(\mathbf{x}_i, y_i) = (h(\mathbf{x}_i) - y_i)^2 = (\mathbf{w}^T \mathbf{x}_i - y_i)^2$
- Two learning algorithms:
  - Stochastic Gradient descend (SGD) / Gradient descend (GD)
  - Pseudo inverse matrix

# Practicalities

- Categorical predictors
- Lost values
- Analyzing residual:
  - Correlation between errors
  - No constant variance of the error
  - Presence of dependence between predictors
- Outliers: (relevant in small samples)

Leer la Sección 3.3.3. del libro : James et al. (2013), “An introduction to Statistical Learning.....”

# Categorical features & Lost values

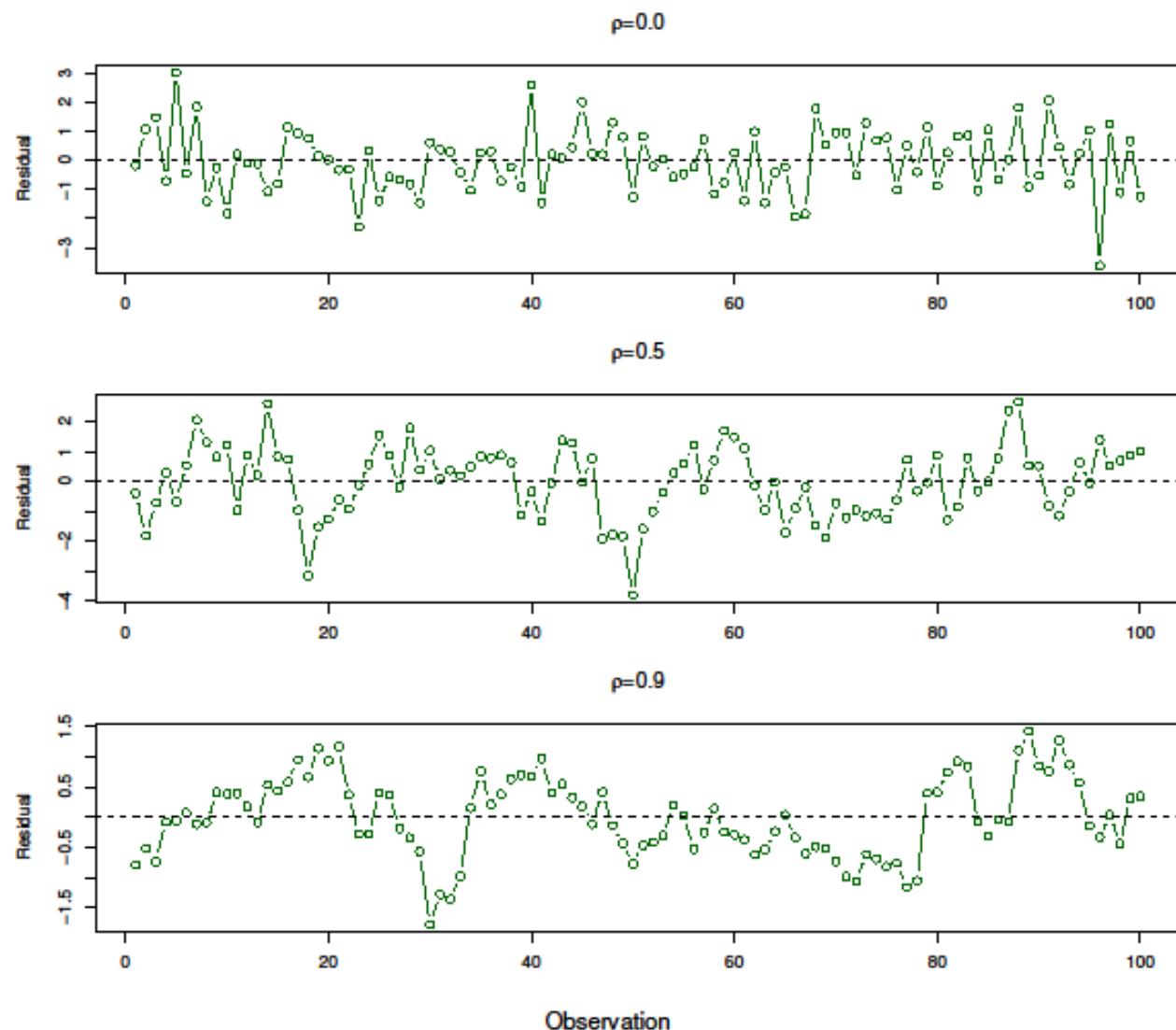
- Categorical features :
  - A categorical feature is coded as an one-hot vector: [0,0,0,...,1,..,0,0] which length is the number categorical values
  - Some non-lineal models can manage them directly: Trees
- Lost values:
  - Some non-lineal models can manage this situation: trees
  - Model using all features:
    - Each value can be interpolated from its feature histogram
    - Each value can be predicted using a specific model

# Correlation in the errors

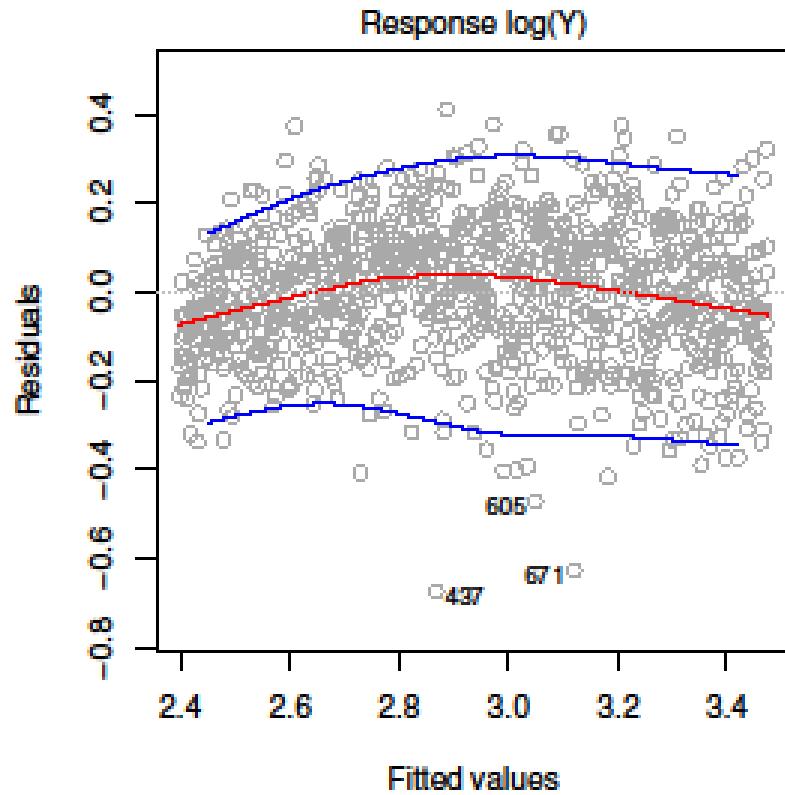
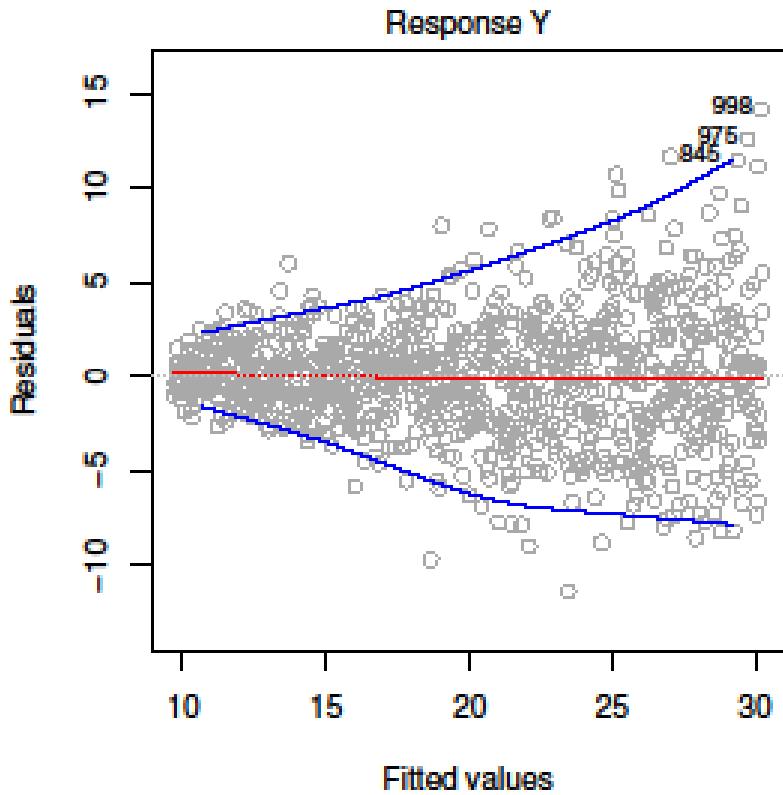
Este problema se presenta cuando **NO se cumple la hipótesis de que los errores de las observaciones son incorrelados.** (**no i.i.d**)

En el estudio de datos asociados a series temporales, es decir datos que explican la conducta de una variable a lo largo del tiempo, es frecuente encontrar este efecto.

También cuando los items de una muestra obtenida para un estudio tienen dependencias no controladas



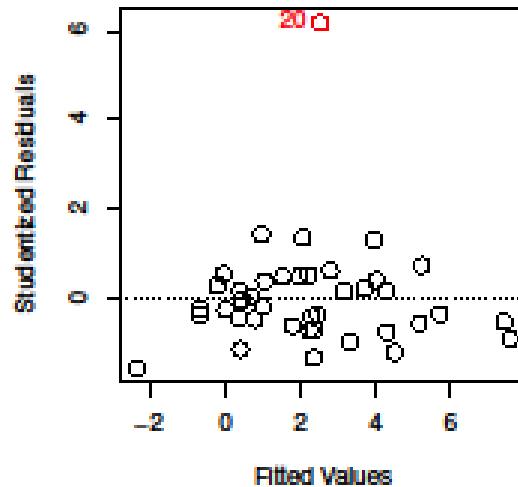
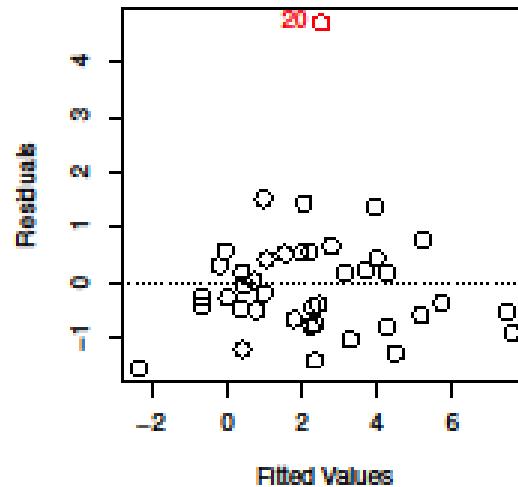
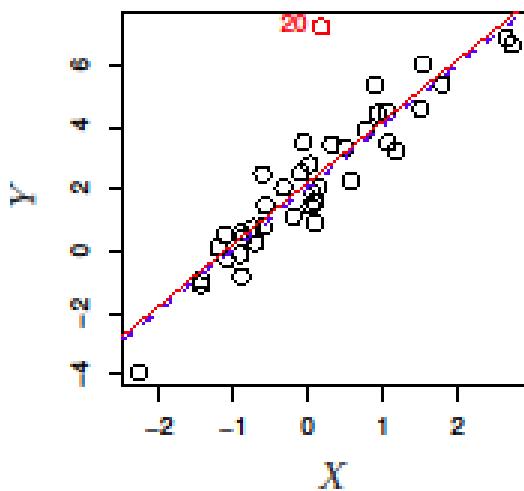
# Non constant variance in the errors



**Heterocedasticidad:** varianzas desiguales en el ruido de las observaciones.

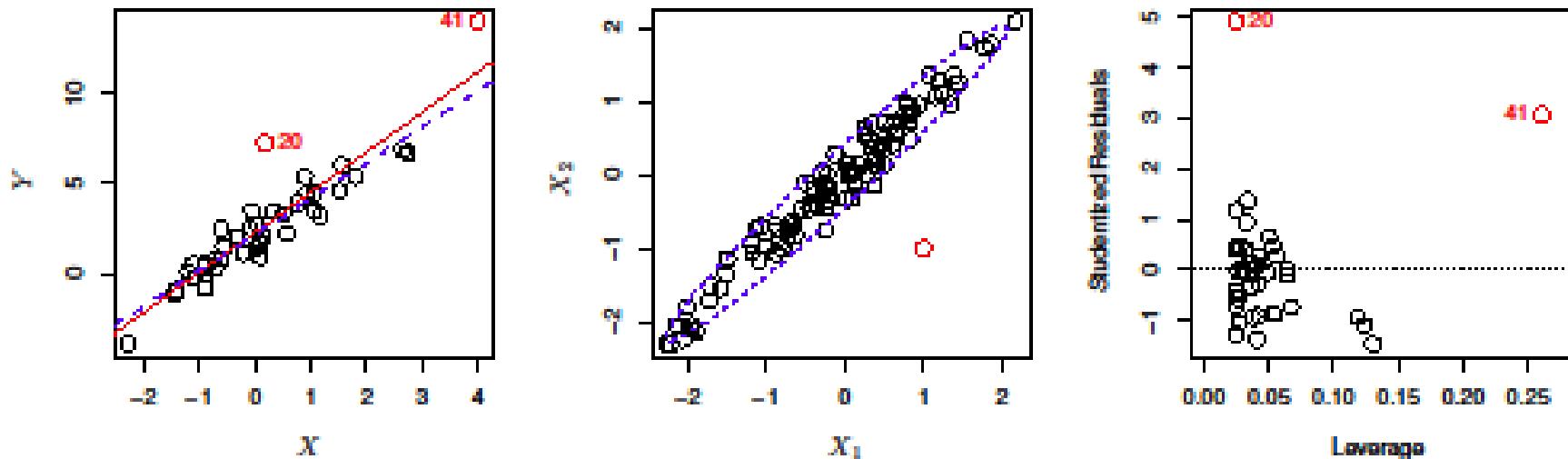
**Tratamiento:** transformar la variable respuesta, o usar Mínimos Cuadrados Ponderados si conocemos una función para la conducta de la covarianza.

# Outliers



- Valores de observación ( $Y$ ) fuera del rango de variabilidad de la muestra
- Si el valor del predictor ( $X$ ) está dentro del rango muestral su influencia en el ajuste será casi nula
- La influencia en el error residual e intervalos de confianza será alta
- Presencia de muchos “outliers” indica que el modelo es inadecuado

# Problema-5: Influencia de los puntos



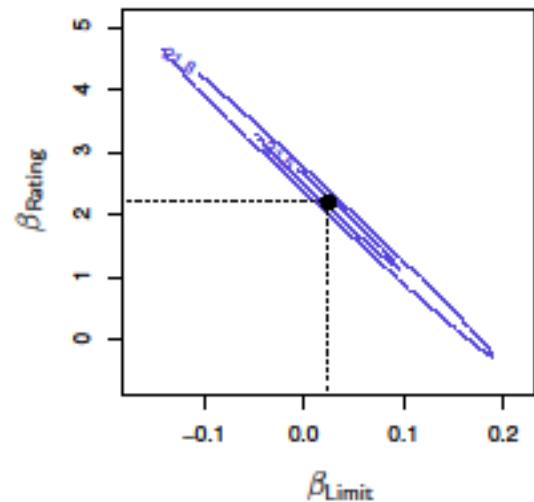
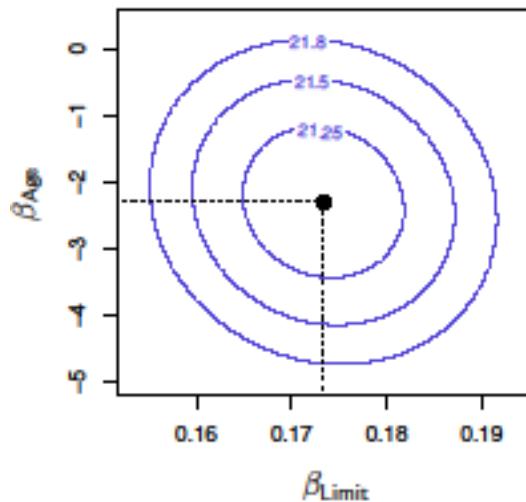
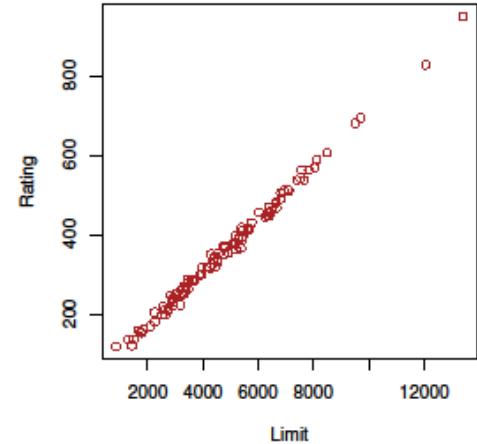
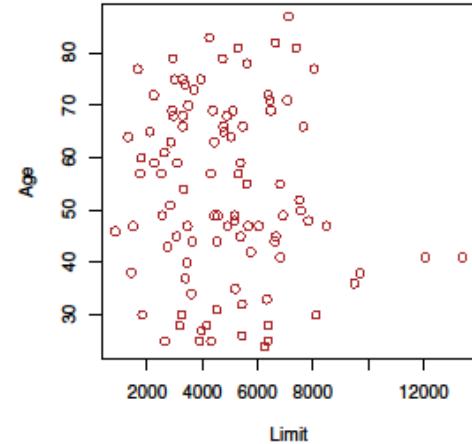
- Los puntos en los extremos del rango de X tienen una alta influencia en el ajuste (izqda)
- En regresión múltiple es posible tener puntos dentro del rango de las variables, pero inusuales a los valores de los predictores (centro)
- **Estadístico-Influencia** en regresión simple:  $h_i = \frac{1}{n} + \frac{(x_i - \bar{x})^2}{\sum_{j=1}^n (x_j - \bar{x})^2} >> \frac{p+1}{n}$ 
  - Ejercicio: calcular usando la Matriz-hat del modelo de regresión lineal simple

# COLLINEARITY BETWEEN PREDICTORS

- Analizamos el problema de que dos o más predictores estén estrechamente relacionados
- La figura arriba-izqda muestra el gráfico de dos variables no relacionadas. Arriba a la drcha dos variables relacionadas
- Las figuras de abajo muestran la conducta de los coeficientes ajustados en función del error residual
  - El punto de mínimo error (negro) obviamente está mucho mejor definido en el caso de la drcha.

• Multicolinealidad:  
• Coeficiente de medida

$$VIF = \frac{1}{1 - R^2_{X_j|X_{-j}}}$$



# Classification

# Data Set

A real data set



# Input Representation

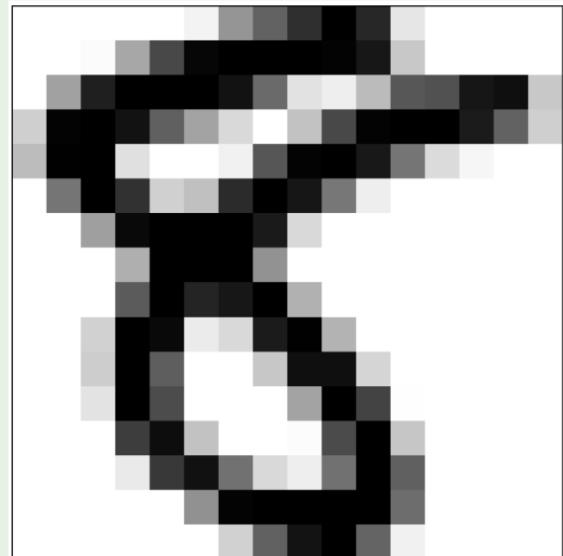
'raw' input  $\mathbf{x} = (x_0, x_1, x_2, \dots, x_{256})$

linear model:  $(w_0, w_1, w_2, \dots, w_{256})$

**Features:** Extract useful information, e.g.,

intensity and symmetry  $\mathbf{x} = (x_0, x_1, x_2)$

linear model:  $(w_0, w_1, w_2)$

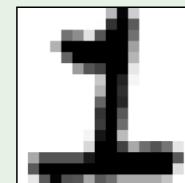
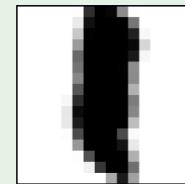
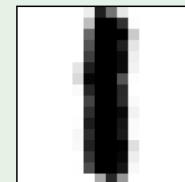
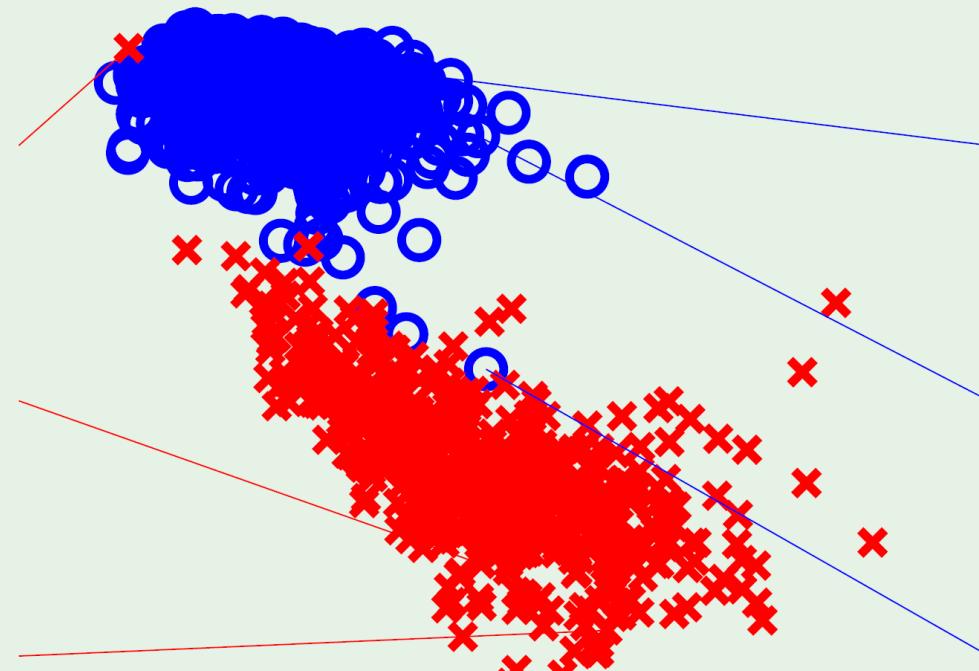
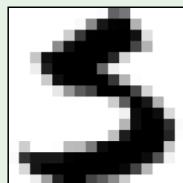
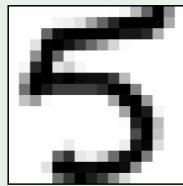
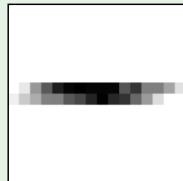


# Illustration of features

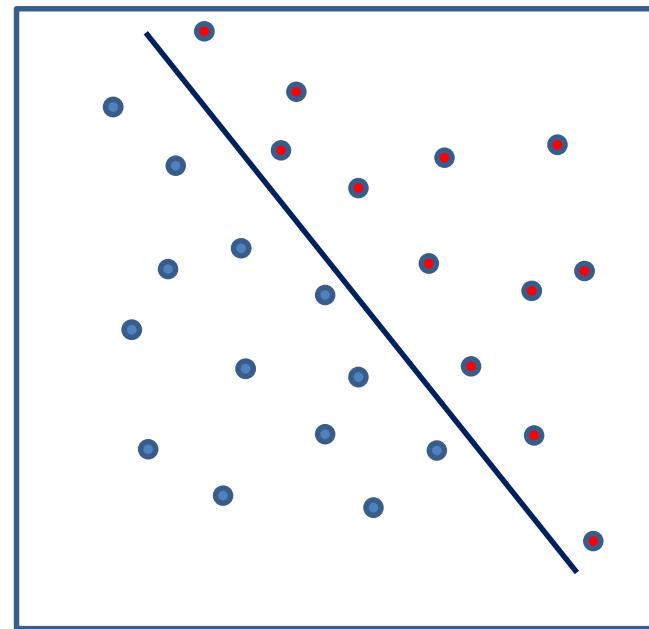
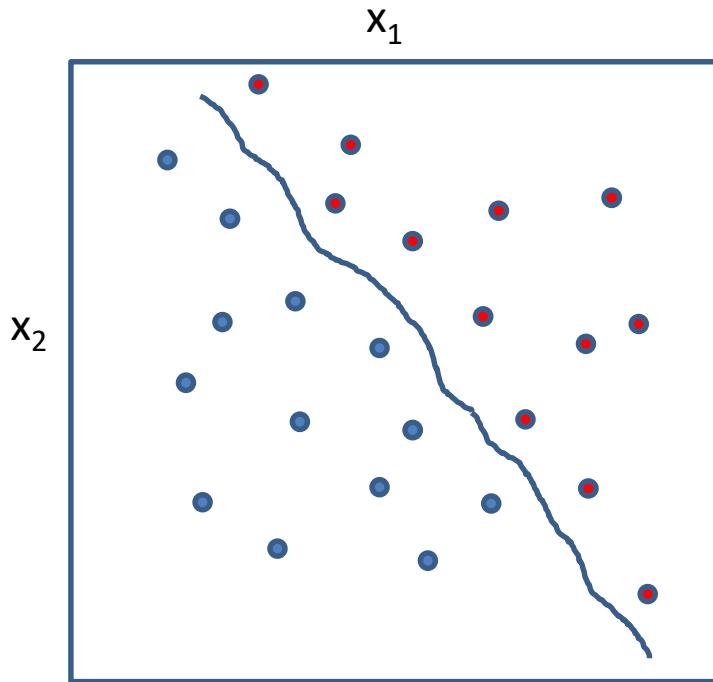
$$\mathbf{x} = (x_0, x_1, x_2)$$

$x_1$ : intensity

$x_2$ : symmetry



# A very simple case



- How to find a separating function ?
- Let assume a very simple class of functions  $\mathcal{H}$  (lines in 2D, hyperplanes in general)
- Let assume a very simple algorithm  $\mathcal{A}$  to search for the best function

# Perceptron

- Assignment Rule:
  - Approve credit / tasty mango: if  $\sum_{i=1}^d x_i w_i \geq b$  (Threshold)
  - Deny credit / non-tasty mango: if  $\sum_{i=1}^d x_i w_i < b$  (Threshold)
- Then,  $h(\mathbf{x}) = \text{sign}\left(\left(\sum_{i=1}^d x_i w_i\right) + b\right)$ ,  $h(\mathbf{x}) \in \{-1, 1\}$ 
  - Equivalently using  $\mathbf{x}^T = (1, x_1, x_2, \dots, x_d, 1)$ ,  $\mathbf{w}^T = (b, w_1, w_2, \dots, w_d)$
  - We can write:
- $\mathcal{H} = \{h \mid h(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x}), \text{ for } \mathbf{w} \text{ fixed}\}$
- This class for  $\mathcal{H}$  is called *Perceptron or lineal separator*
- Learning Criteria (ERM) :  $\min_{\mathbf{w} \in \mathbb{R}^{d+1}} \underbrace{\frac{1}{N} \sum_{n=1}^N [\![\text{sign}(\mathbf{w}^T \mathbf{x}_n) \neq y_n]\!]}_{E_{in}}$
- *How to minimize it ?*

# Perceptron Learning Algorithm (PLA)

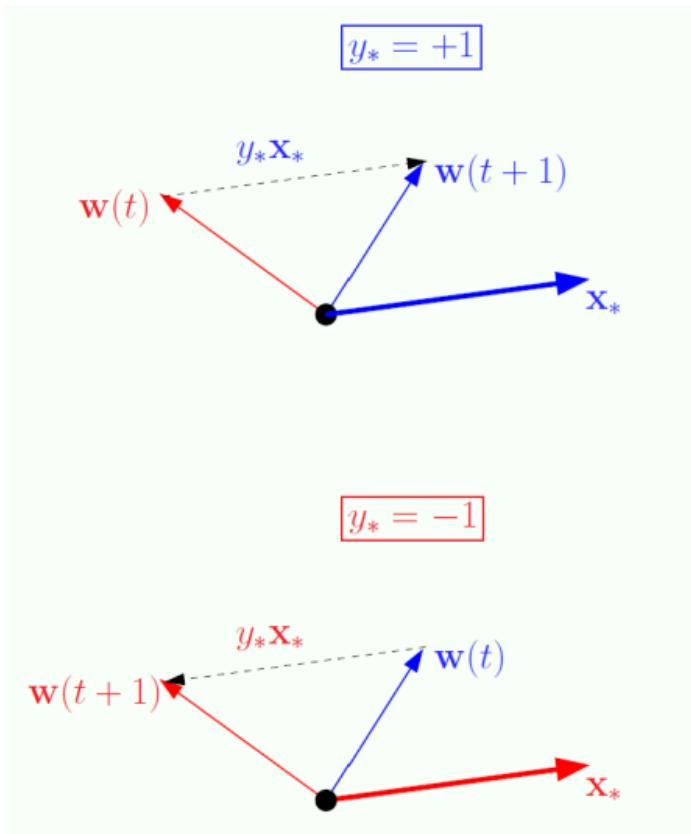
- Simple algorithm searching for the best,  $h \in \mathcal{H}$ , through iterative evaluations of the training data ( $\mathcal{D}$ ):

LEARNING RULE :

$$\begin{cases} w_{updated} = w_{current} + y \cdot x & \text{if } sign(w^T x_i) \neq y_i \\ w_{updated} = w_{current} & \text{if } sign(w^T x_i) = y_i \end{cases}$$

Connection to G.D. later on

- The learning algorithm iterates on the data sample but memoryless
- For linearly separable data set:
  - The PLA algorithm is always able to find a vector  $\hat{w}$  such that  $h(x_i) = y_i$  for all the training samples ( $\mathcal{D}$ ) in finite time .
  - PLA learns an optimal rule using data only !!



# Linear Classification Algorithm

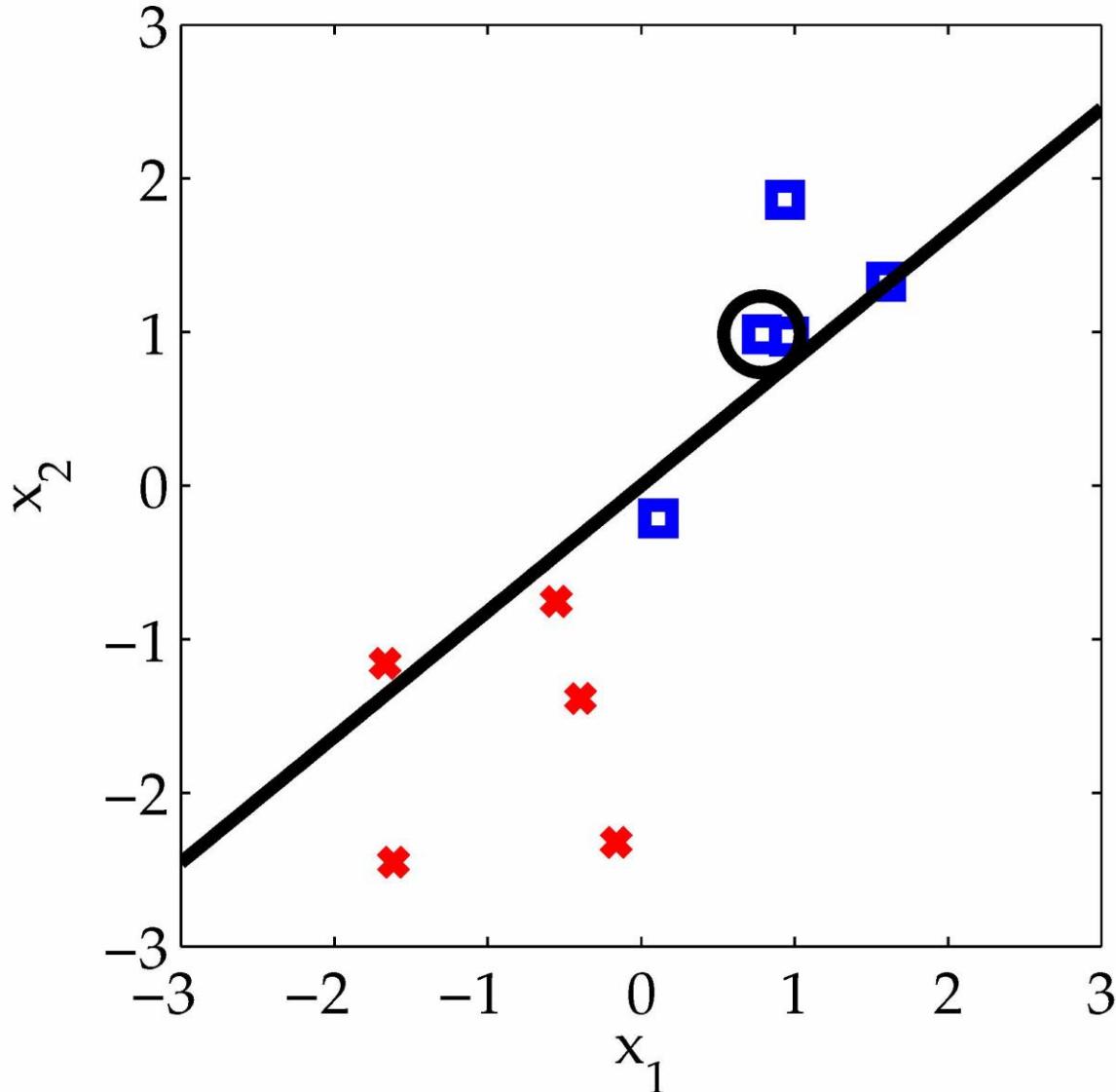
- Linear Perceptron:
  - Given the data set  $(\mathbf{x}_n, y_n), n = 1, 2, \dots, N$
  - Step.1: Fix  $\mathbf{w}_{\text{ini}} = 0$
  - Step.2: Iterate on the  $\mathcal{D}$ -samples improving the solution:
  - repeat
    - For each  $x_i \in \mathcal{D}$  do
    - if:  $\text{sign}(\mathbf{w}^T \mathbf{x}_i) \neq y_i$  then
      - update  $\mathbf{w}$ :  $\mathbf{w}_{\text{new}} = \mathbf{w}_{\text{old}} + y_i \mathbf{x}_i$
    - else continue
    - End for
  - Until No changes in a full pass on  $\mathcal{D}$

# Remarkable Properties

- **The learning algorithm iterates on the data sample but memoryless**
  - On each update the PLA fix only one item of the training sample without paying attention to the others, but very remarkably the algorithm reaches the optimum in finite time.
- **For linearly separable data set:**
  - The PLA algoprithm is always able to find a vector  $\hat{w}$  such that  $h(x_i) = y_i$  for all the training samples ( $\mathcal{D}$ ) in finite time .
- **Clearly, the PLA learns an optimal rule using data only ! !**

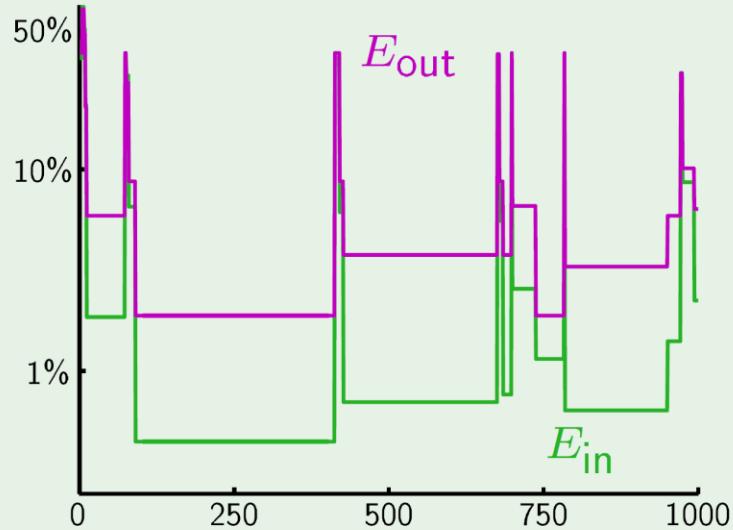
# How the perceptron works

Iteration 1, Instance 1

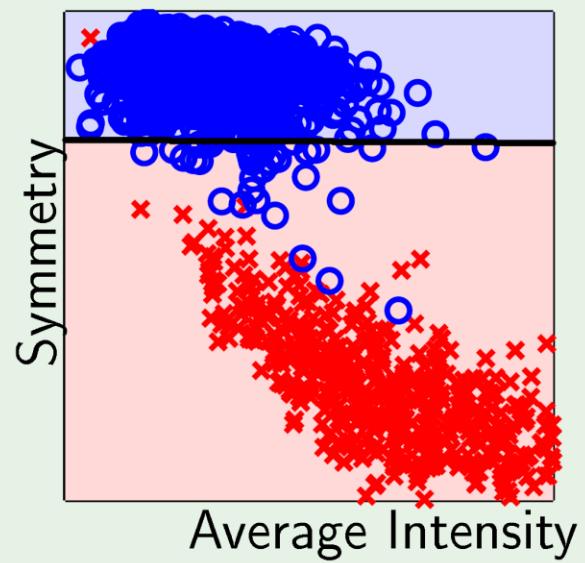


# What PLA does

Evolution of  $E_{\text{in}}$  and  $E_{\text{out}}$



Final perceptron boundary



# Linear Classification

- Let now assume  $\mathcal{X} = \{1\} \times \mathbb{R}^d$ ,  $\mathcal{Y} = \{-1, +1\}$  and  $h: \mathcal{X} \rightarrow \mathcal{Y}$
- The linear model for binary classification uses the hypothesis class  $\mathcal{H}$ ,

$$h(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x})$$

where  $\mathbf{w} \in \mathbb{R}^{d+1}$ ,  $d$  being the dimensionality of the input space and  $x_0 = 1$

- Questions to answer:
  - Q1. Can we make sure that  $E_{out}(g) - E_{in}(g) \approx 0$  ?
  - Q2. Can we make  $E_{in}(g) \approx 0$  ?
- Q1 : According to the learning theory we get

$$E_{out}(g) \leq E_{in}(g) + \mathcal{O}\left(\sqrt{d \frac{\log N}{N}}\right)$$

As we can see the generalization error is comparable to that of linear regression(  $\mathcal{O}\left(\frac{d}{N}\right)$ )

Then for  $N$  sufficiently large  $E_{out}$  and  $E_{in}$  will be very close to each other

# Linear Classification: $E_{in}(g) \approx 0$

- We consider two scenarios:
  1. Linearly separable Data : which means there exist  $w^*$  with  $E_{in}(w^*)=0$
  2. Non-Separable data
- Scenary-1:
  - We already saw the Perceptron Learning Algorithm (PLA) (remember?)
- PLA-Result: Let  $(x_1, y_1), \dots, (x_m, y_m)$  be a separable sample of examples. Let  $B = \min\{\|w\|\}: \forall i \in [m], y_i w^T x_i \geq 1\}$  and let  $R = \max_i \|x_i\|$ . Then the PLA stops after at most  $(RB)^2$  iterations, and when it stops it holds that  $\forall i \in [m], y_i w^T x_i \geq 0$ .
- The convergence rate depends on the parameter B. But this value can be exponentially large in d.
- As far as PLA is concerned linear separability is a property of the data  $\mathcal{D}$  not the target  $f$ .
  - A linearly separable  $\mathcal{D}$  could have been generated either from a linearly separable target, or (by chance) from a non-separable target

# Non-separable data: $E_{in}(g) \approx 0$ ?

Non-Separable data: Two different scenarios:

- NS.1: Errors on a linear separable solution
- NS.2: There not exists a linear separable solution

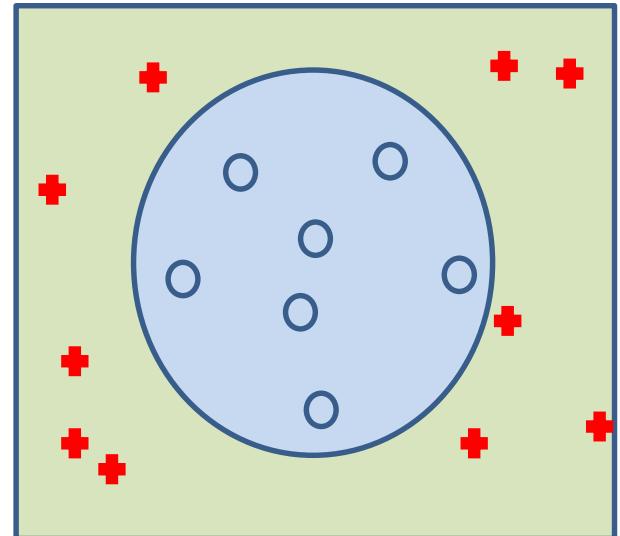
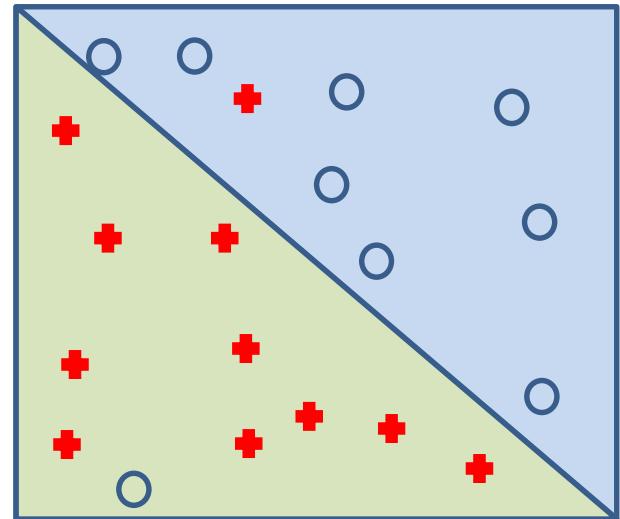
To find a solution in the NS.1 we have to solve

$$\underbrace{\min_{w \in R^{d+1}} \frac{1}{N} \sum_{n=1}^N \llbracket \text{sign}(w^T x_n) \neq y_n \rrbracket}_{E_{in}}$$

But this problem is known to be a combinatorial problem with NP-complexity

We will approximate the problem with the Pocket Algorithm

NS.2 is approached using Non-Linear transforms (later on)

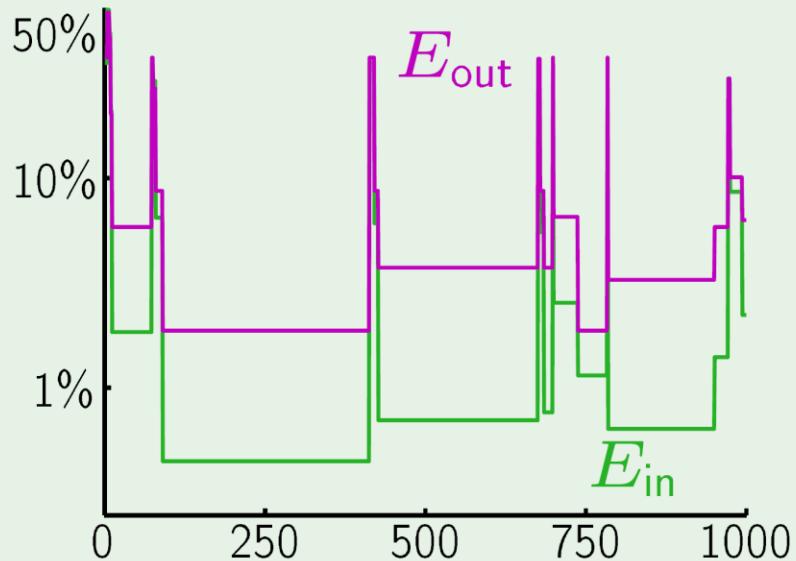


# Non-Separable Case: An approximation

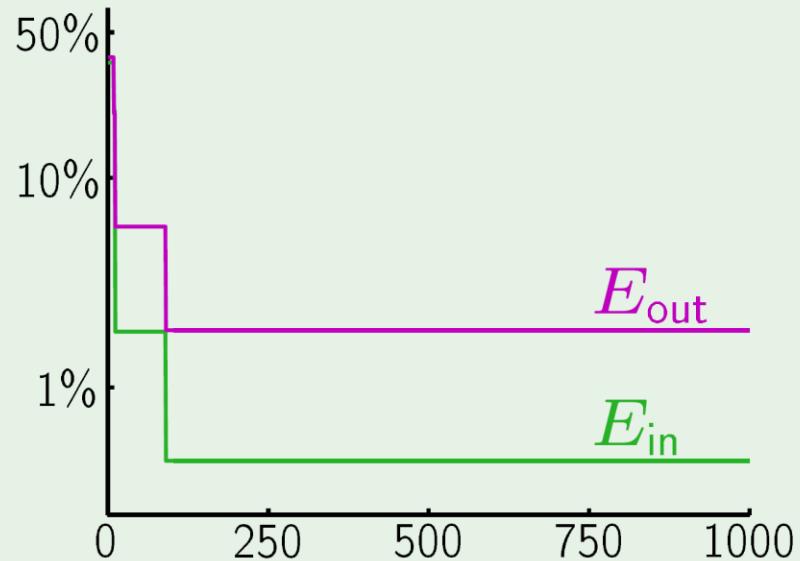
- The Pocket Algorithm: essentially the pocket algorithm keeps “in its pocket” the best vector solution encountered up the iteration t in PLA
- POCKET ALGORITHM:
  1. Set the pocket weight vector  $\hat{w}$  to  $w(0)$  of PLA
  2. **for**  $t=1, \dots, T$  **do**
  3. Run PLA for one update to obtain  $w(t+1)$
  4. Evaluate  $E_{in}( w(t+1) )$
  5. If  $w(t+1)$  is better than  $\hat{w}$  in terms of  $E_{in}( w(t+1) )$ , set  $\hat{w} = w(t+1)$
  6. **Return**  $\hat{w}$
- The pocket algorithm has a clear efficiency penalty in point 4.
- But it is guaranteed to get a good solution after a fixed large number of updatings.

# The ‘Pocket’ Algorithm

PLA:

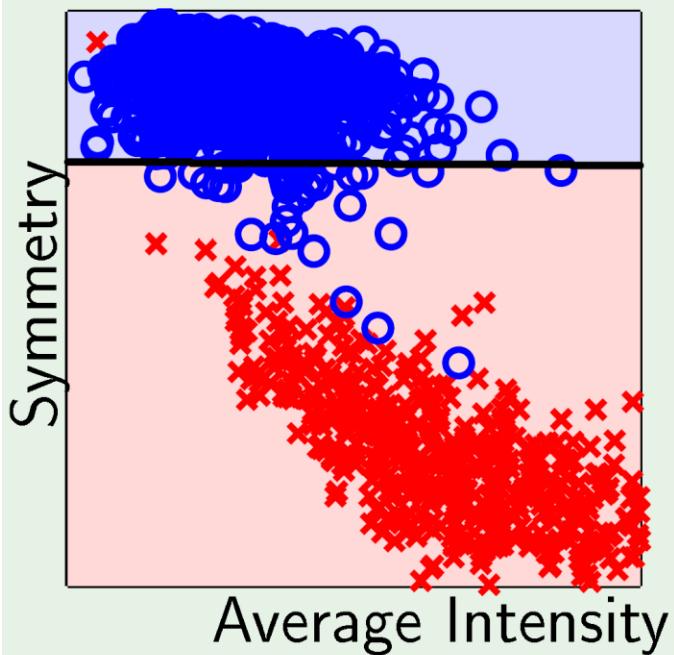


Pocket:

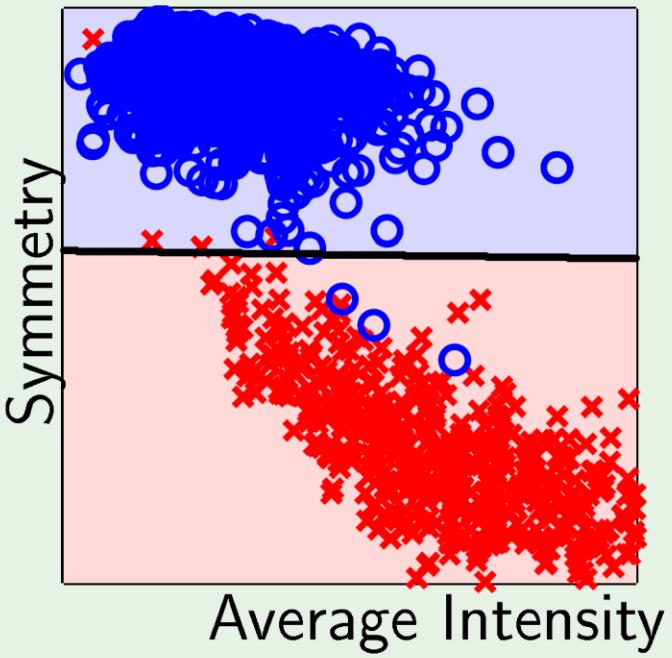


# Classification boundary – The PLA vs Pocket

PLA:



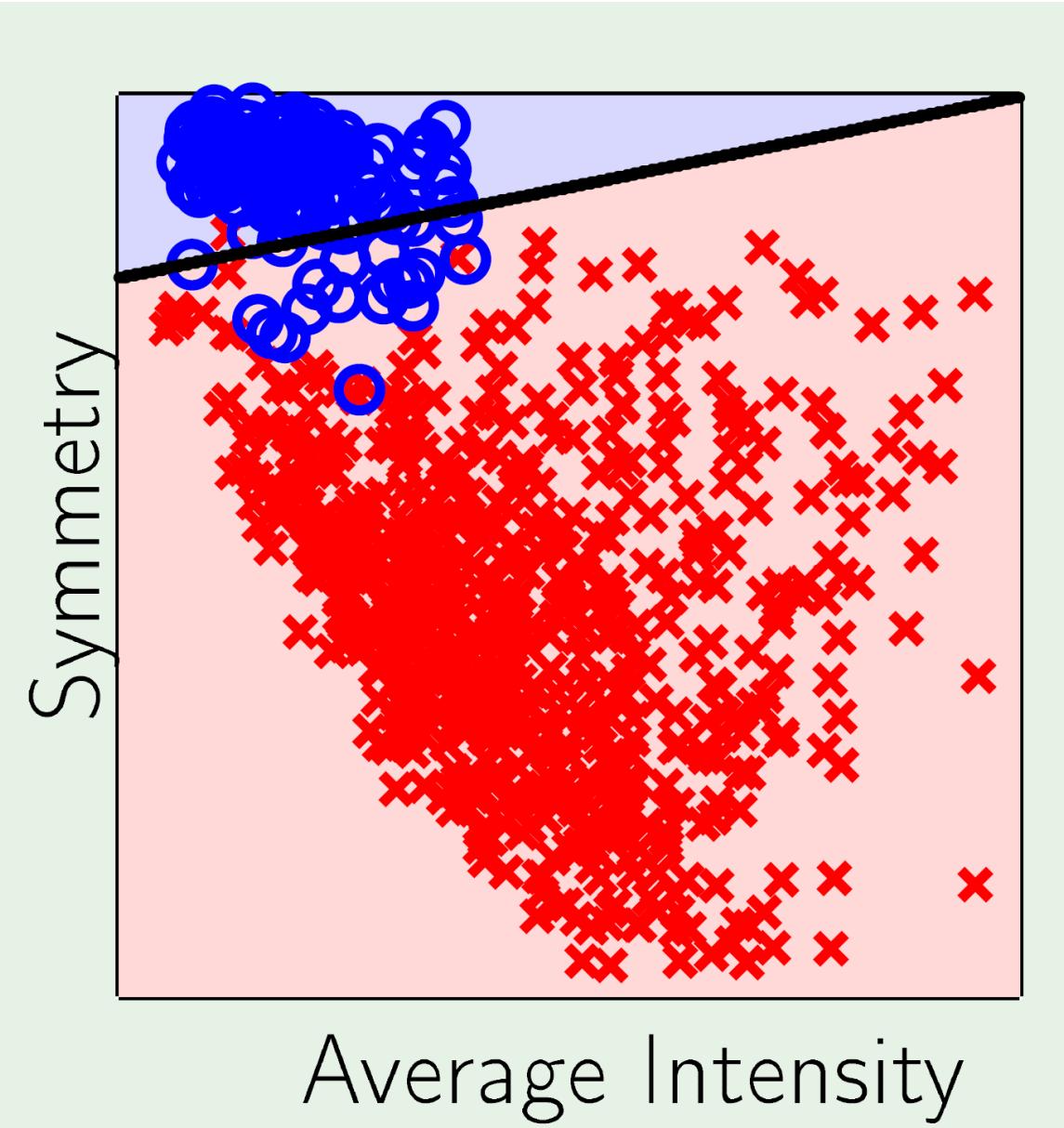
Pocket:



# Linear Regression for Classification

- Linear regression learns a real function  $y = f(\mathbf{x}) \in \mathbb{R}$
- Binary-valued functions are also real valued!  $\pm 1 \in \mathbb{R}$
- Use linear regression to get  $\mathbf{w}$  where  $\mathbf{w}^T \mathbf{x}_n \approx y_n = \pm 1$
- In this case the  $\text{sign}(\mathbf{w}^T \mathbf{x}_n)$  is likely to agree with  $y_n = \pm 1$
- Good initial weights for classification !

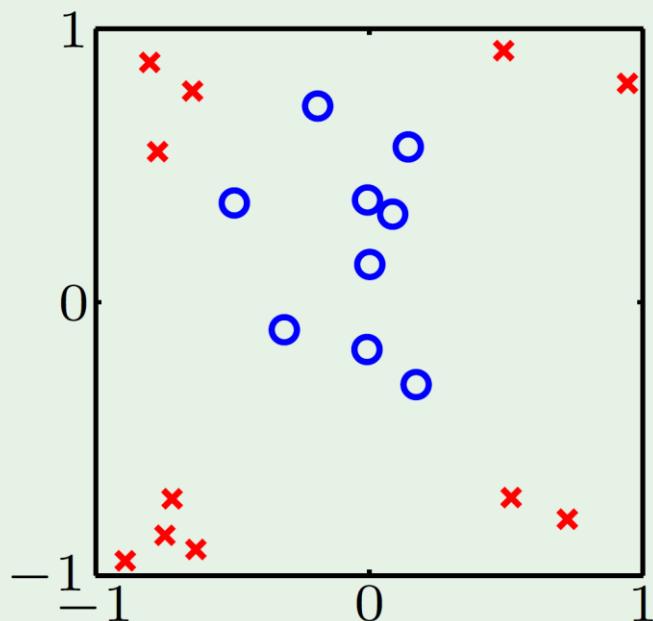
# Linear Regression Boundary



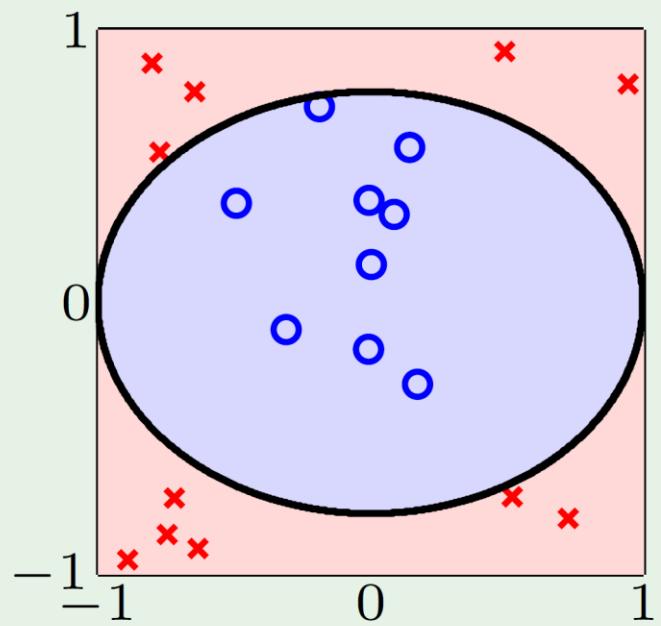
# Adding functions to H: Non linear Transformations

# Linearity in predictors has limitations

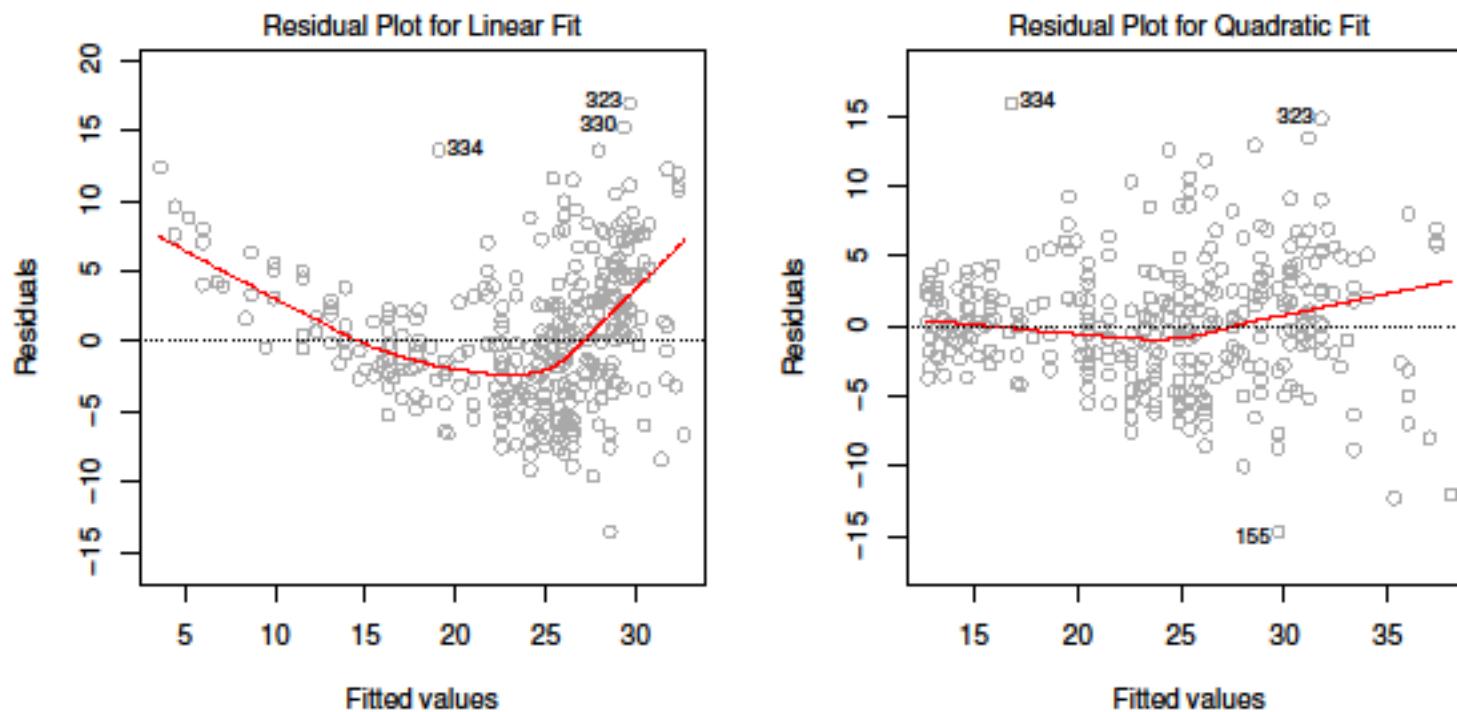
Data:



Hypothesis:



# Regression: Non linear predictors



Left plot : A linear feature model shows a clear non linear behaviour on the residual  
Right plot : A quadratic feature model improves the residual behaviour

# Linear Model means linearity in $w$

- Credit Example: Credit line is affected by “years of residence” but NOT in a linear way
  - Nonlinear  $\llbracket x_i > 5 \rrbracket$  and  $\llbracket x_i < 1 \rrbracket$  are better features

How can we do this with linear models ?

- Linear regression implement:

$$\sum_{i=0}^d x_i w_i$$

- Linear classification implement:

$$\text{sign}\left(\sum_{i=0}^d x_i w_i\right)$$

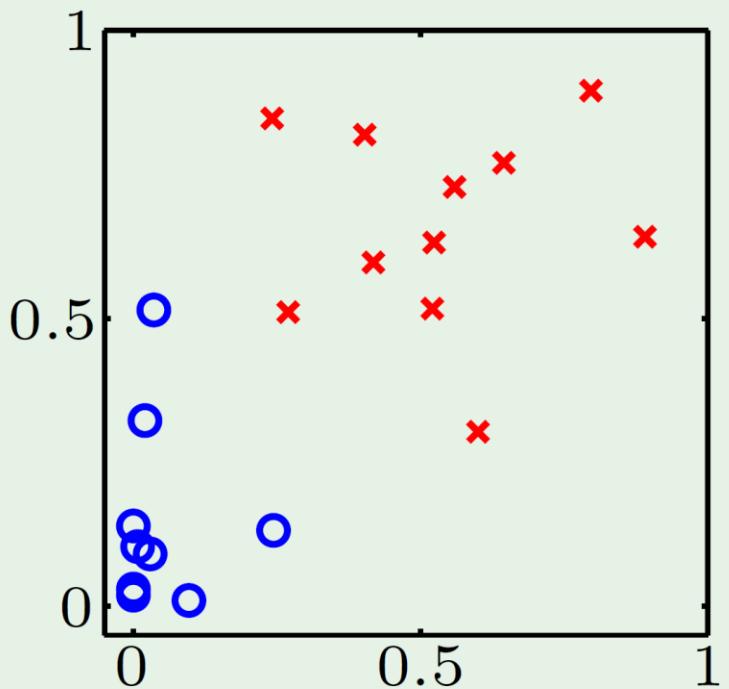
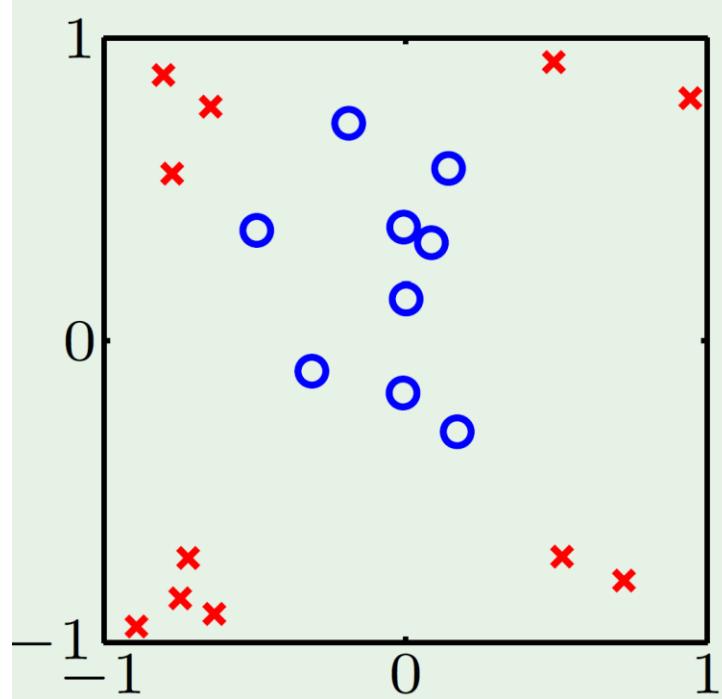
Algorithms work because the **linearity in the weights**

# NonLinear Transformations(NLT)

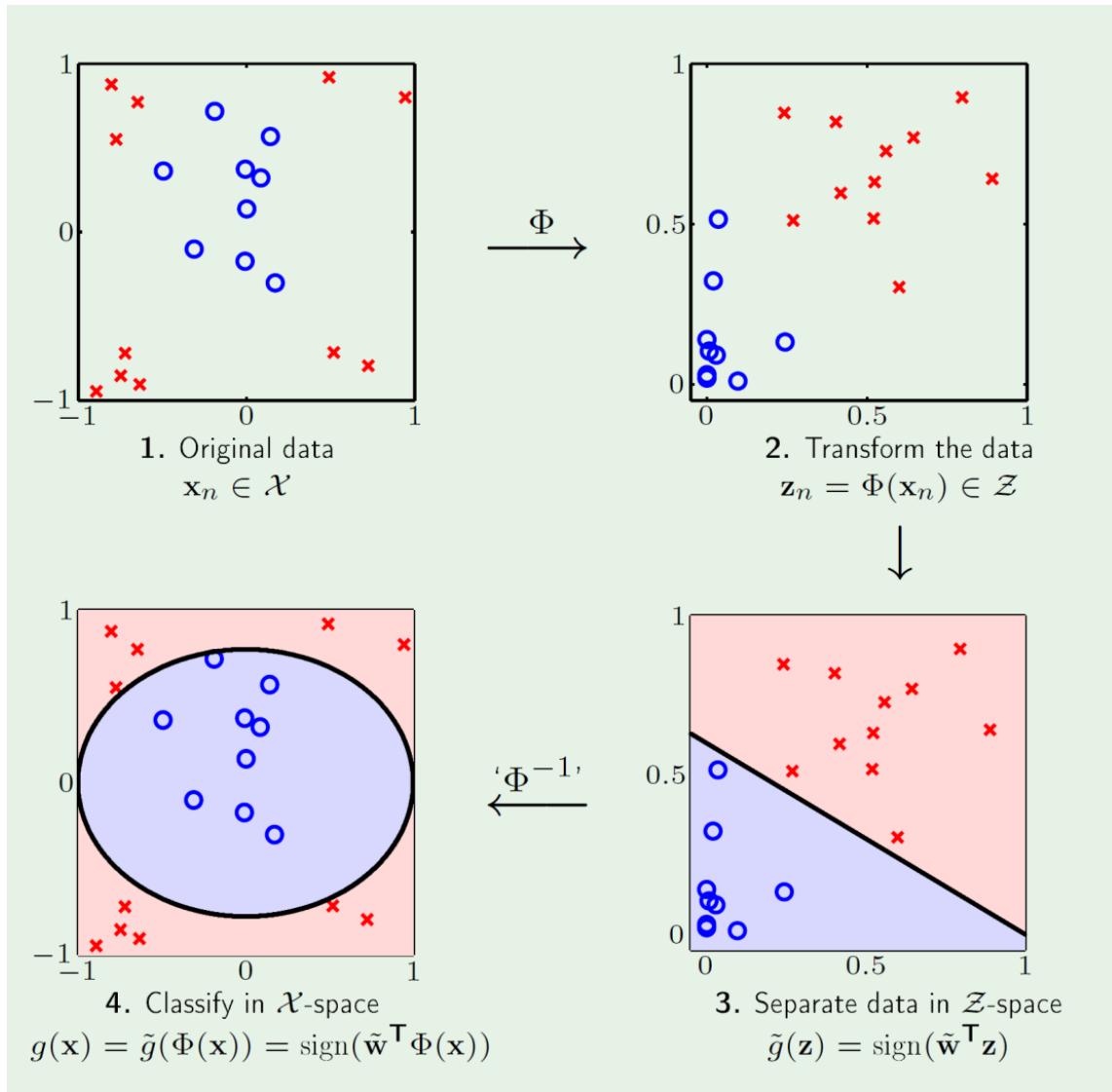
- A linear model only has to be linear in  $w$
- Therefore we can use non-linear functions of the  $x_i$  and still remaining inside of the linear model class.
  - Polinomial functions:  $x^2, x^3, \dots, x^n$ ,
  - Transcendent functions:  $\log(x), \sin(x), x^{1/2}, \dots,$
  - Boolean functions:  $[x > 1] \& [x < 5], [x < 1] \& [x > 5]$ , etc
  - etc
- The use of NLT does NOT change the class of hypothesis  $\mathcal{H}$  BUT the original  $\mathcal{X}$ -space is transformed to the  $\mathcal{Z}$ -space including non-linear features.
- Nonlinear transformations should be used when:
  - The residual error after the fitting with the original features shows non-linear behaviour
  - We know some features are non-linear functions of the others
  - BUT NOT ONLY !!

# Transform the data nonlinearly

$$(x_1, x_2) \xrightarrow{\Phi} (x_1^2, x_2^2)$$



# Nonlinear Transformations



# What transforms to what

$$\mathbf{x} = (x_0, x_1, \dots, x_d) \xrightarrow{\Phi} \mathbf{z} = (z_0, z_1, \dots, z_{\tilde{d}})$$

$$\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N \xrightarrow{\Phi} \mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_N$$

$$y_1, y_2, \dots, y_N \xrightarrow{\Phi} y_1, y_2, \dots, y_N$$

No weights in  $\mathcal{X}$

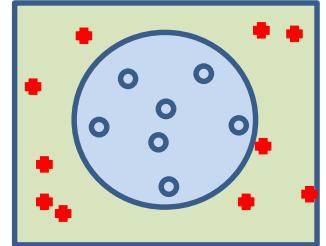
$$\tilde{\mathbf{w}} = (w_0, w_1, \dots, w_{\tilde{d}})$$

$$g(\mathbf{x}) = \text{sign}(\tilde{\mathbf{w}}^\top \Phi(\mathbf{x}))$$

# NLT- Example

- Let consider the classification problem given in the figure
- Let assume the non-linear function  $x_1^2 + x_2^2 = 0.6$

represents the separating curve between the two classes (inside and outside the circle).



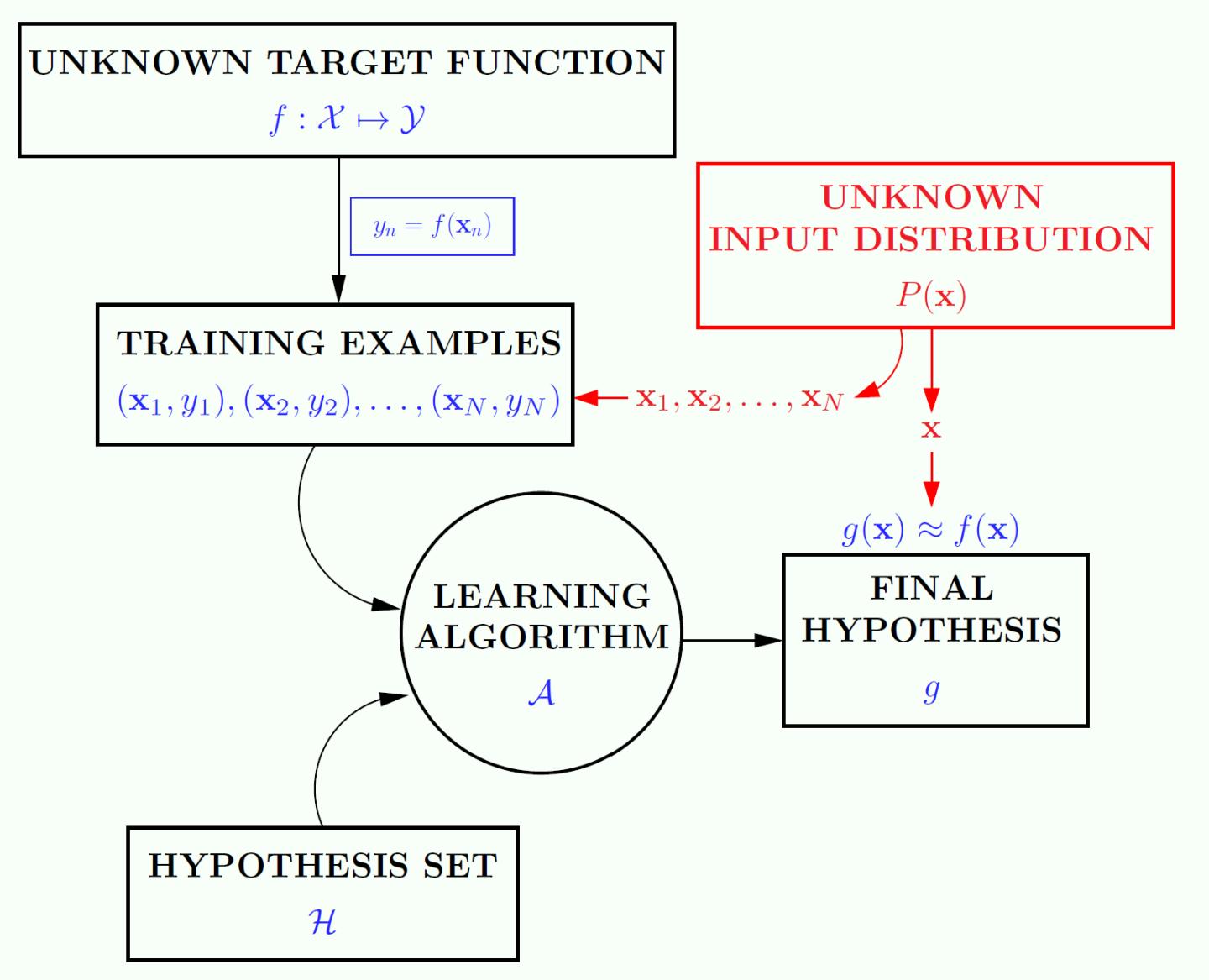
- Then we know that the hypothesis  $h(\mathbf{x}) = \text{sign}((1) x_1^2 + (1)x_2^2 - 0.6)$  solves the problem, **BUT** this hypothesis can NOT be implemented by the regular PLA

$$(x_1, x_2, 1) \xrightarrow{\Phi} (x_1^2, x_2^2, 1) = (z_1, z_2, 1)$$

- Let write the solution using  $\mathbf{z}$ :  $\tilde{h}(\mathbf{z}) = \text{sign}((1) z_1 + (1)z_2 - 0.6)$  Fine!!  
$$\tilde{h}(\mathbf{z}) = \tilde{h}(\Phi(\mathbf{x})) = h(\mathbf{x})$$
- In real problems we have to **fix the transform before seeing the data**. So, we do not know the better specific transforms.

# Error Measures and Noisy Data: A learning setup generalization

# The current Learning Setup



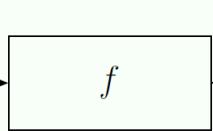
# ERM: A Learning Rule

- The Goal: Minimize the sample error in order to choose a final hypothesis with very low error out of the sample.
- **Empirical Risk Minimization (ERM): examples**
  - Classification problems:  $ERM_{\mathcal{H}}(\mathcal{D}) = \operatorname{argmin}_h \left\{ \frac{1}{N} \sum_{i=1}^N I[y_i \neq h(\mathbf{x}_i)] \right\}$
  - Regression problems:  $ERM_{\mathcal{H}}(\mathcal{D}) = \operatorname{argmin}_h \left\{ \frac{1}{N} \sum_{i=1}^N (y_i - h(\mathbf{x}_i))^2 \right\}$
- Two cases:
  - Simple case:  $f$  is a deterministic UNKNOWN function ( here,  $y_i = f(x_i)$ )
  - Real case:  $f$  is a stochastic UNKNOWN function given by  $P(Y | X)$  ( later on)

$$ERM_{\mathcal{H}}(\mathcal{D}) \in \operatorname{argmin}_{h \in \mathcal{H}} E_{in}(h)$$

- When  $f \in \mathcal{H}$ , is expected that  $ERM_{\mathcal{H}}(\mathcal{D}) \leq \varepsilon$

# Error measure user specified



$$\begin{cases} +1 & \text{you} \\ -1 & \text{intruder} \end{cases}$$

Two types of error.

		$f$	
		+1	-1
$h$	+1	no error	<b>false accept</b>
	-1	<b>false reject</b>	no error

In any application you need to think about how to penalize each type of error.

		$f$
		+1    -1
$h$	+1	0    1
	-1	<b>10</b> 0

Supermarket

		$f$
		+1    -1
$h$	+1	0 <b>1000</b>
	-1	1    0

CIA

## ***Take Away***

*Error measure is specified by the user.*

*If not, choose one that is*

- plausible (conceptually appealing)
- friendly (practically appealing)

# Noisy Samples

- A realistic scenario should consider noisy LABELINGS, this is:

$$y_n \neq f(\mathbf{x}_n)$$

- In general we have **Noisy Targets** due to lack of information on the variables defining the unknown function
- **In other words the target function not always a deterministic function**
- Credit approval case:
  - Two customer with identical data can have different answer
- Tasty mangos:
  - Items with the same features can be labeled as of different class

# Target distribution

- Instead of  $y = f(x)$  we now use

$$P(y|x)$$

- Each sample  $(x,y)$  is generated from the joint distribution

$$P(x)P(y|x)$$

- **Noisy target** is a deterministic target  $f(x) = E(y|x)$  plus noise  $y - f(x)$

- **Deterministic target** is a special case of noisy target

$P(y|x)$  is zero except in  $y = f(x)$

# Connection between $P(Y|X)$ and $P(X)$

- Both distribution are relevant in the learning process
- The target distribution  $P(y|x)$  is what we want to learn
- The input distribution  $P(x)$  quantify the relative importance of  $x$
- Let's consider the expression  $P(x,y) = P(x)P(y | x) = P(x | y)P(y)$ 
  - Clearly, learning is over once we know  $P(x,y)$
- There are two ways of computing  $P(y|x)$ 
  - To learn  $P(y|x)$  directly (Discriminative approaches )
  - To learn  $P(y|x)$  through  $P(x | y)$  using the Bayes rule (Generative approaches )

# The Bayes rule

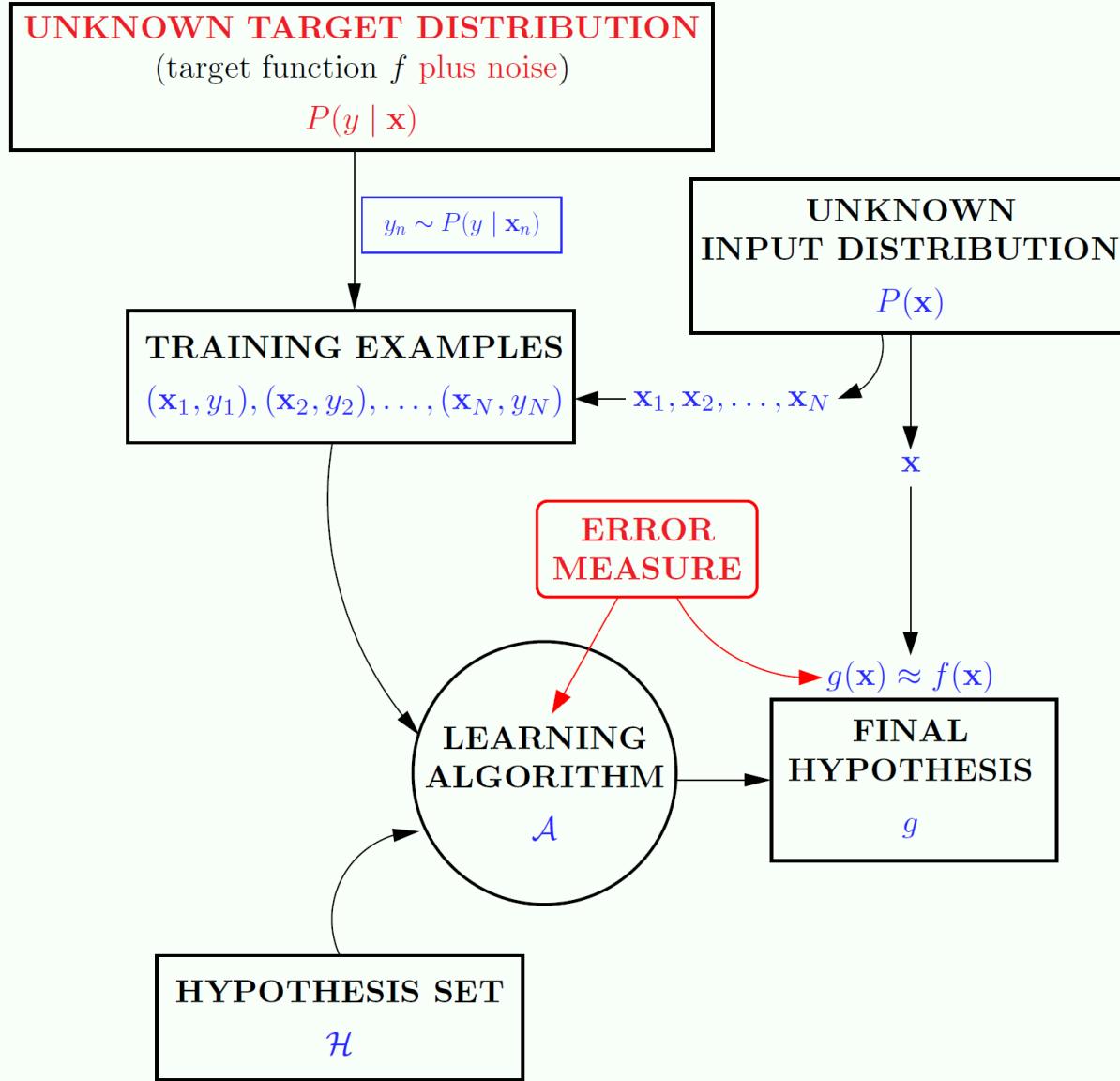
- A relevant question to ask is: if we knew the true probability distribution  $\mathcal{P}$ , could we generate an optimum hypothesis (the minimum risk hypothesis) for any learning task?
- Answer: Given any probability distribution  $\mathcal{P}$  over  $X \times \{0,1\}$ , the best label predicting function from  $X$  to  $\{0,1\}$  will be

$$f_{\mathcal{P}}(x) = \begin{cases} 1 & \text{if } \mathbb{P}[y = 1|x] \geq 1/2 \\ 0 & \text{otherwise} \end{cases} \quad \text{Bayes rule}$$

- It is not difficult to show that for any probability distribution does NOT exist other function with lower risk than  $f_{\mathcal{P}}$ , that is for any other function  $g$ ,  $E_{out}(f_{\mathcal{P}}) \leq E_{out}(g)$
- Unfortunately, we do NOT know the probability distribution  $\mathcal{P}$
- An alternative approach to function learning is to estimate  $\mathcal{P}$  from the samples and classifying with the Bayes Rule (GENERATIVE MODELS)

“To solve a problem never solve a more complex one as intermediate step” (Vapnik 1995).

# Learning Setup with Error Measure and Noisy Target



# PROBABILITY ESTIMATION

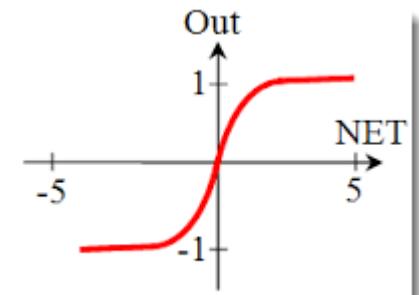
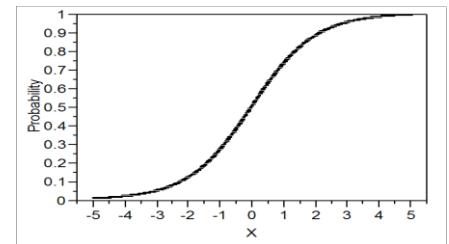
# Logistic Regression (LGR)

- Linear Regression:  $h(x) = \mathbf{w}^T \mathbf{x}$
- Linear Classification:  $h(x) = \text{sign}(\mathbf{w}^T \mathbf{x}) \in \{-1,1\}$  or  $\{0,1\}$
- **Logistic Regression:**  $h(x) \in [0,1]$  or  $h(x) \in [-1,1]$ 
  - Logistic Regression is a binary classifier based on the strength of the regression model.
- Some advantages appears:
  - The LGR output can be considered as a probability on the classification.
  - It allows us more flexibility in order to assign samples to the labels
- The  $\sigma$ -function is called *logistic function (sigmoide)*:

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$

- Other function:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



# Logistic Regression vocabulary

- The ratio  $\frac{h(\mathbf{x})}{1-h(\mathbf{x})}$  is called “odds”
- $\ln(\text{odds}) = \ln\left(\frac{h(\mathbf{x})}{1-h(\mathbf{x})}\right) = \mathbf{w}^T \mathbf{x}$  (función logit)
- The logit function can be any regression model.
  - It is therefore possible to define decision boundaries from nonlinear function of the variables
- $$\frac{h(\mathbf{x})}{1-h(\mathbf{x})} = e^{\mathbf{w}^T \mathbf{x}} \quad \xrightarrow{\hspace{2cm}} \quad h(\mathbf{x}) = \frac{e^{\mathbf{w}^T \mathbf{x}}}{1+e^{\mathbf{w}^T \mathbf{x}}} = \sigma(\mathbf{w}^T \mathbf{x}) ,$$
$$1 - h(\mathbf{x}) = \frac{1}{1+e^{\mathbf{w}^T \mathbf{x}}} = \sigma(-\mathbf{w}^T \mathbf{x})$$

# Error Measure

- Formally, we are trying to learn the target function  $f(\mathbf{x}) = \mathbb{P}[y = +1|x]$   
But the data does not give the values of  $f(\mathbf{x})$  *explicitely* but values generated by a noisy target  $y$

$$P(y|\mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{for } y = +1 \\ 1 - f(\mathbf{x}) & \text{for } y = -1 \end{cases}$$

Given an hypothesis  $h$ , how close is from  $f$  in terms of the noisy target ?

$$P(y|\mathbf{x}) = \begin{cases} h(\mathbf{x}) & \text{for } y = +1 \\ 1 - h(\mathbf{x}) & \text{for } y = -1 \end{cases}$$

Since  $h(\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x})$  then  $P(y|\mathbf{x}) = \sigma(y \mathbf{w}^T \mathbf{x})$

Since the data points are independents, the probability of the sample is

$$L(\mathbf{w}) = \prod_{i=1}^N P(y_i|\mathbf{x}_i) = \prod_{i=1}^N \sigma(y_i \mathbf{w}^T \mathbf{x}_i)$$

# Fitting Criteria

Maximum Likelihood (ML): To choose the hypothesis  $h$  which maximize  $L(\mathbf{w})$

Equivalently Minimize ERM:

$$E_{\text{in}}(\mathbf{w}) = -\frac{1}{N} \ln(L(\mathbf{w})) = \frac{1}{N} \sum_{i=0}^N \ln\left(\frac{1}{P(y_i|\mathbf{x}_i)}\right) = \frac{1}{N} \sum_{i=0}^N \underbrace{\ln(1 + e^{-y_i \mathbf{w}^T \mathbf{x}_i})}_{\mathbf{e}(h(x_i), y_i)}$$

We can see that  $\mathbf{e}(h(x_i), y_i)$  is small when  $y_i \mathbf{w}^T \mathbf{x}_i \gg 0$

$$\begin{aligned}\nabla_{\mathbf{w}} E_{\text{in}}(\mathbf{w}) &= \frac{\partial}{\partial \mathbf{w}} \left( \frac{1}{N} \sum_{i=0}^N \ln(1 + e^{-y_i \mathbf{w}^T \mathbf{x}_i}) \right) = \frac{1}{N} \sum_{i=0}^N -y_i \mathbf{x}_i \frac{e^{-y_i \mathbf{w}^T \mathbf{x}_i}}{1 + e^{-y_i \mathbf{w}^T \mathbf{x}_i}} \\ &= \frac{1}{N} \sum_{i=0}^N -y_i \mathbf{x}_i \sigma(-y_i \mathbf{w}^T \mathbf{x}_i)\end{aligned}$$

# SGD+ERM: A new induction rule

- We can apply SGD to solve any optimization problem where the gradient can be computed.
- Example.1 : Regression
  - The error function is  $e_n(\mathbf{w}) = (y_n - \mathbf{w}^T \mathbf{x}_n)^2$
  - Assume the size of the sample is  $O(10^8)$ , and the number of features is  $O(10^3)$ . In this case it is not feasible to compute the matrix inverse.
  - Fortunately, SGD can solve the problem iterating on bunch of examples.
  - Furthermore, linear regression convergence can be speeded using the second derivative and the **Newton method**.
- Example.2 : Perceptron (PLA)
  - In PLA we minimize the error according to the updating rule:  $\mathbf{w}(t+1) = \mathbf{w}(t) + y_i \mathbf{x}_i$
  - This rule can be seen as a SGD rule with  $\eta=1$  and gradient given by  $y_i \mathbf{x}_i$
  - Can be verified that the function  $e_n(\mathbf{w}) = \max(0, -y_n \mathbf{w}^T \mathbf{x}_n)$  provides the gradient
- Example.3 : Logistic Regression (LGR)
  - Now the error function is  $e_n(\mathbf{w}) = \ln(1 + e^{-y_n \mathbf{w}^T \mathbf{x}_n})$
  - For  $\mathbf{w}$  large  $e_n(\mathbf{w}) \approx \max(0, -y_n \mathbf{w}^T \mathbf{x}_n)$  making LGR+SGD equivalent to PLA

# Multilabel classification

- Binary logistic regression can be generalized to cover k labels
  - In the binary approach we maximize the probability of the binary labelling given by the likelihood of N Bernoulli variables
- For K labels we applied the same methodology BUT now the likelihood is defined using N multi-Bernoulli variables (i.e. variables con k outputs)
  - The labels , denoted by  $y$  are represented by 1-of-K vector:  $[0,0,\dots,1,\dots,0]$  or  $[-1,-1,\dots,1,\dots,-1]$

Example: Let's assume the binary case ( $N=2$ ) with labels  $\{0,1\}$

- Let  $X$  be a single Bernoulli random variable :  $P(X=1)=p$  and  $P(X=0)=q$ ,  $p+q=1$
- $P(X)=p^X q^Y$ ,  $Y=1-X$ ,  $X \in \{0,1\}$
- $P(x_1, x_2, \dots, x_N) = P(x_1) \times P(x_2) \times \dots \times P(x_N) = p^{\sum x_i} q^{\sum y_i} = p^{\#[x=1]} q^{\#[x=0]}$

# The sample likelihood

Let  $(\mathbf{x}, \mathbf{y})$  denote a single sample, then

$$P(\mathbf{y}|\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_k) = \prod_{k=1}^K P(y_k | \mathbf{w}_k, \mathbf{x}) = \prod_{k=1}^K \sigma(w_k^T \mathbf{x})^{y_k}$$

Let  $\{\mathbf{X}, \mathbf{Y}\}$  denote a sample of N-items

$$P(\mathbf{Y}|\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_k) = \prod_{n=1}^N \prod_{k=1}^K P(y_{nk} | \mathbf{w}_k, \mathbf{x}_n)$$

- The likelihood is defined by

$$L(\mathbf{Y}|\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_k) = \prod_{n=1}^N \prod_{k=1}^K \left( \sigma(w_k^T \mathbf{x}_n) \right)^{y_{nk}}$$

# Multilabel classification: SoftMax

- $t_{nk} = \sigma(\mathbf{w}_k^T \mathbf{x}_n)$ ,  $\sum_{k=1}^K t_{nk} = 1$ ,
- $E(\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_K) = -\ln L(Y | \mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_K) = -\sum_{n=1}^N \sum_{k=1}^K y_{nk} \ln t_{nk}$
- $\nabla_{\mathbf{w}_j} E(\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_K) = \sum_{n=1}^N (t_{nj} - y_{nj}) \mathbf{x}_n$
- **SOFMAX:** from each sample  $\mathbf{x}$  a vector of probabilities is computed using the estimated vector  $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_K$

$$P(C_j | \mathbf{x}) = \frac{\exp(\mathbf{w}_j^T \mathbf{x})}{\sum_k \exp(\mathbf{w}_k^T \mathbf{x})}, \quad j = 1, \dots, K$$

**Decision Rule:** To assign  $\mathbf{x}$  to  $C_j$  if  $P(C_j | \mathbf{x}) = \max_k P(C_k | \mathbf{x})$ ,  $k = 1, \dots, K$

# Understanding $E_{\text{out}}$

# Bias-Variance Tradeoff

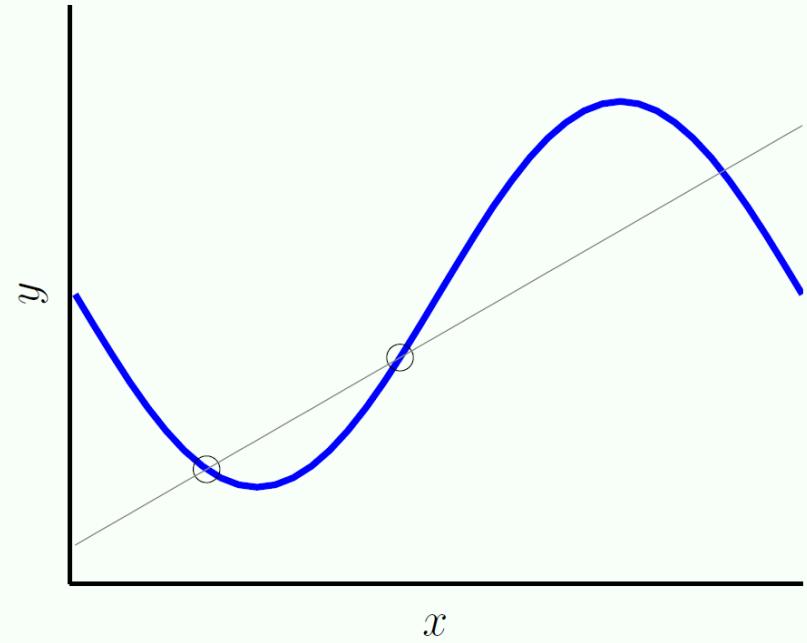
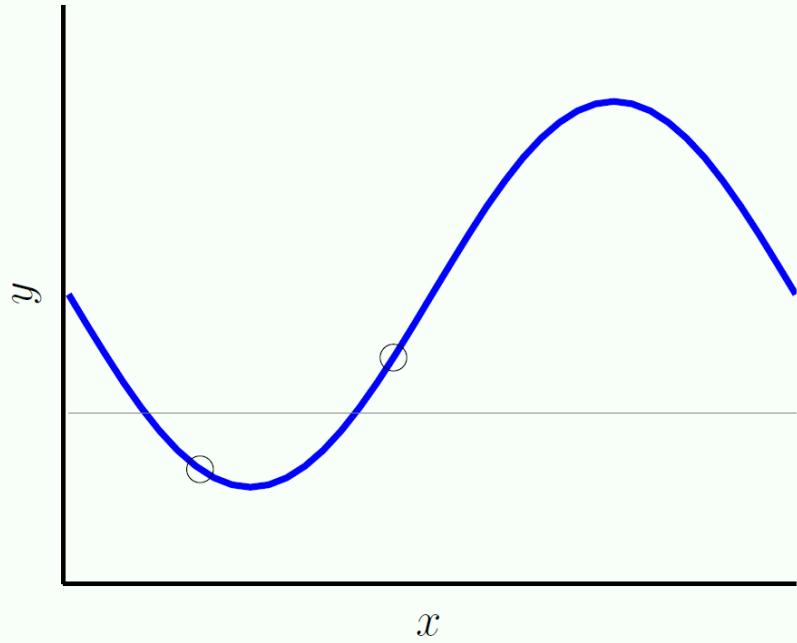
- BIAS-VARIANCE decomposition

$$E_{\text{out}}(g^{(\mathcal{D})}) = \mathbb{E}_x \left[ (g^{(\mathcal{D})}(x) - f(x))^2 \right]$$

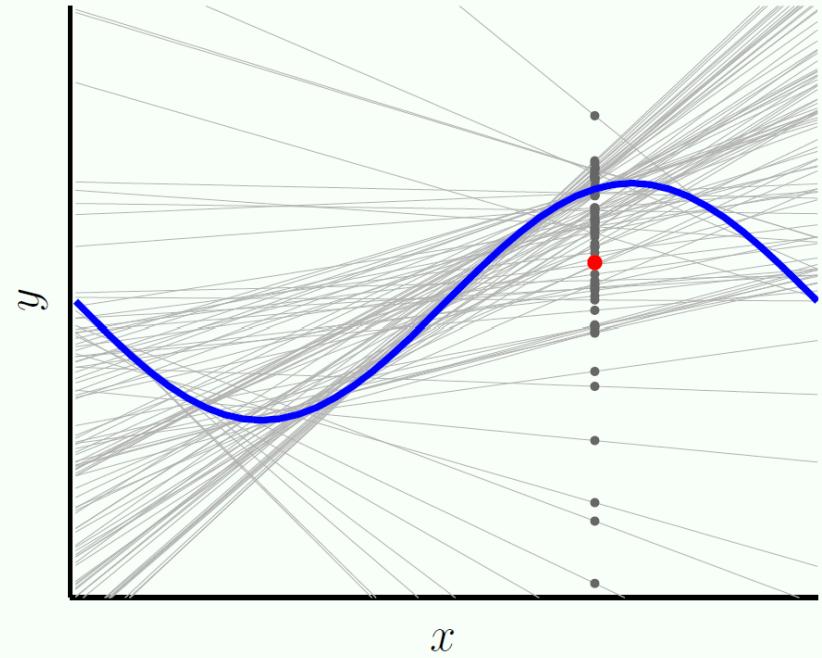
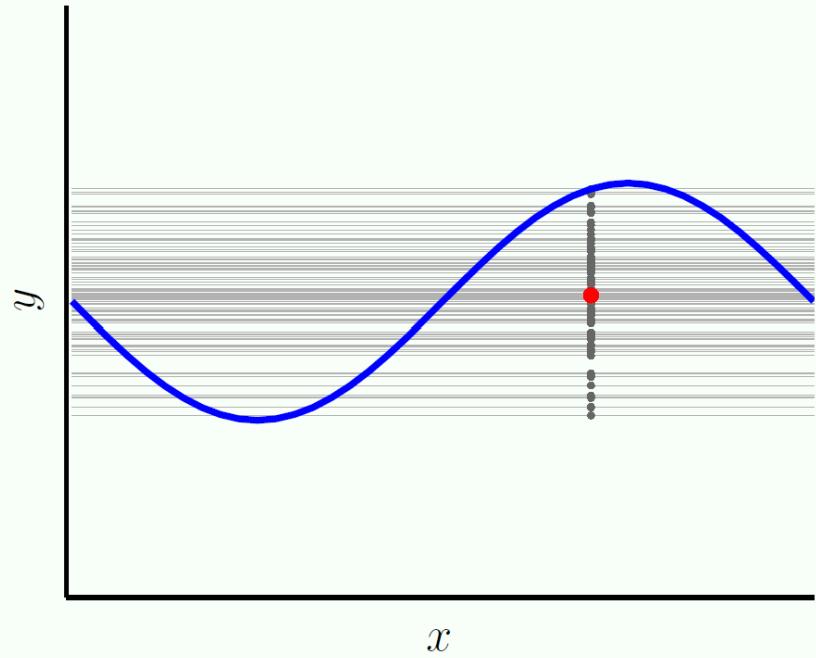
- $\mathbb{E}_x$  denotes the expected value with respect to  $x$  ( based on  $\mathbb{P}(\mathcal{X})$ )
- That is the Mean Squared Error (MSE) of  $g^{(\mathcal{D})}$
- Bias-variance analysis applies to squared errors (classification and regression)
- Bias-variance analysis can take into account the learning algorithm
- Different learning algorithms can have different  $E_{\text{out}}$  when applied to the same  $\mathcal{H}!!$

# A simple learning problem

- 2 data points. 2 hypothesis sets
- $\mathcal{H}_0: h(x) = b$
- $\mathcal{H}_1: h(x) = ax + b$

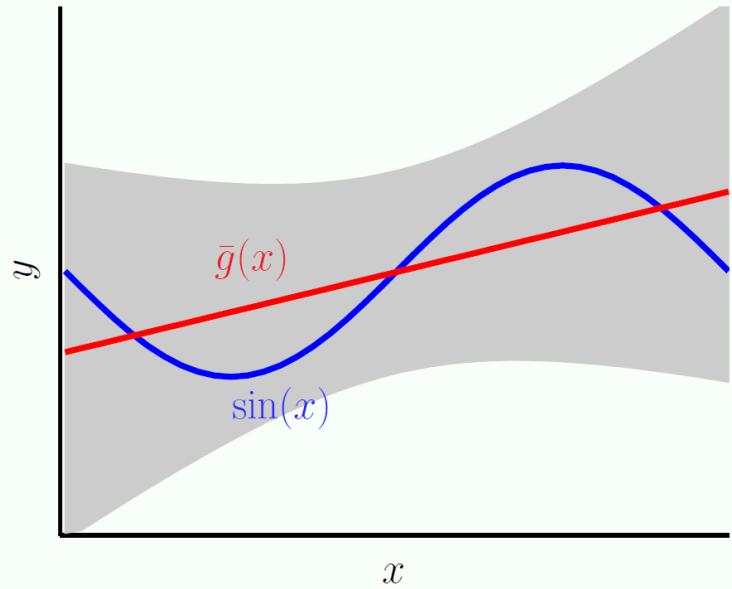
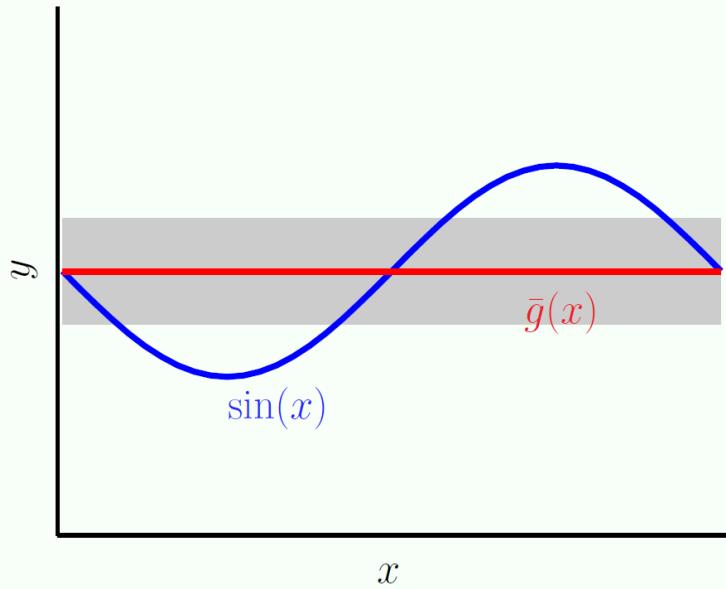


# Let repeat the experiment multiples times ...



- For each data set  $\mathcal{D}$ , you get a different  $g^{\mathcal{D}}$ .
- So, for a fixed  $\mathbf{x}$ ,  $g^{\mathcal{D}}(\mathbf{x})$  is random value, depending on  $\mathcal{D}$ .

# What's Happening on Average?



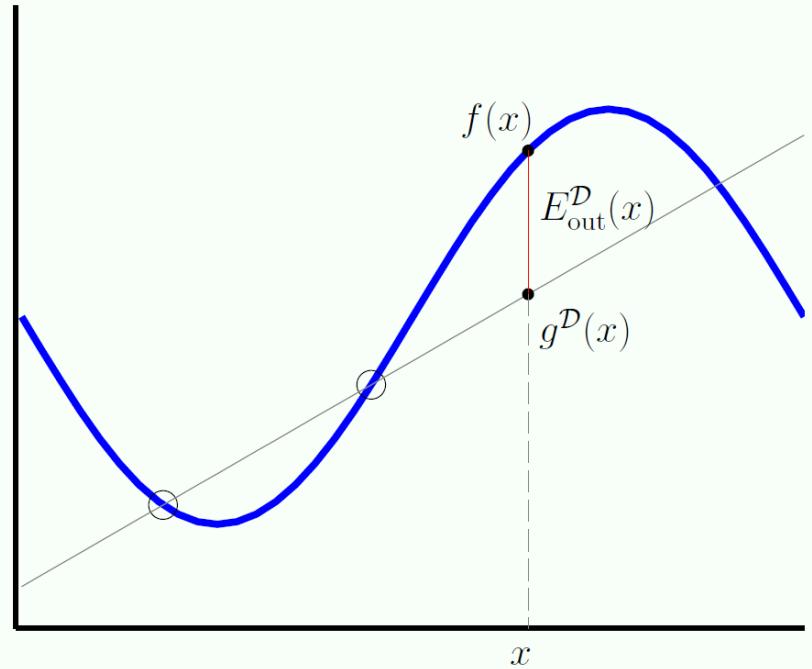
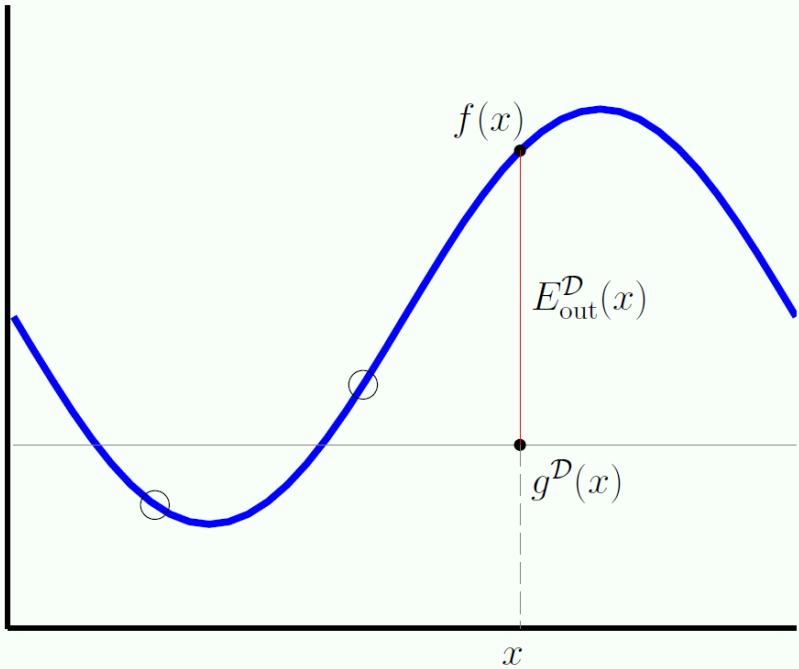
We can define

$$g^{\mathcal{D}}(\mathbf{x}) \quad \leftarrow \text{random value, depending on } \mathcal{D}$$

$$\begin{aligned} \bar{g}(\mathbf{x}) &= \mathbb{E}_{\mathcal{D}}[g^{\mathcal{D}}(\mathbf{x})] \\ &\approx \frac{1}{K}(g^1(\mathbf{x}) + g^2(\mathbf{x}) + \dots + g^K(\mathbf{x})) \quad \leftarrow \text{the average prediction on } \mathbf{x} \end{aligned}$$

$$\begin{aligned} \text{var}(\mathbf{x}) &= \mathbb{E}_{\mathcal{D}}[(g^{\mathcal{D}}(\mathbf{x}) - \bar{g}(\mathbf{x}))^2] \\ &= \mathbb{E}_{\mathcal{D}}[g^{\mathcal{D}}(\mathbf{x})^2] - \bar{g}(\mathbf{x})^2 \quad \leftarrow \text{how variable is the prediction?} \end{aligned}$$

## $E_{out}$ on Test Point $\mathbf{x}$ for Data $\mathcal{D}$



$$E_{out}^D(\mathbf{x}) = (g^D(\mathbf{x}) - f(\mathbf{x}))^2 \quad \leftarrow \text{squared error, a random value depending on } \mathcal{D}$$

$$E_{out}(\mathbf{x}) = \mathbb{E}_{\mathcal{D}}[E_{out}^D(\mathbf{x})] \quad \leftarrow \text{expected } E_{out}(\mathbf{x}) \text{ before seeing } \mathcal{D}$$

# Bias-Variance Tradeoff

- In order to get an estimation of the MSE error independent of  $\mathcal{D}$

$$\mathbb{E}_{\mathcal{D}}[E_{out}(g^{(\mathcal{D})})] = \mathbb{E}_{\mathcal{D}}\left[\mathbb{E}_x\left[(g^{(\mathcal{D})}(x) - f(x))^2\right]\right] = \mathbb{E}_x\left[\mathbb{E}_{\mathcal{D}}\left[(g^{(\mathcal{D})}(x) - f(x))^2\right]\right]$$

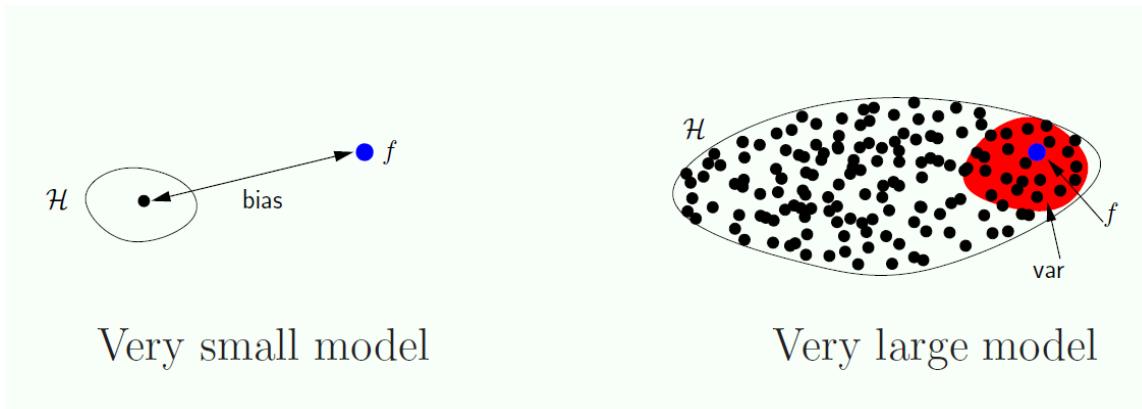
$$\mathbb{E}_{\mathcal{D}}\left[(g^{(\mathcal{D})}(x) - f(x))^2\right] = \mathbb{E}_{\mathcal{D}}(g^{\mathcal{D}}(x)^2) - 2 \mathbb{E}_{\mathcal{D}}(g^{(\mathcal{D})}(x)f(x)) + f(x)^2$$

- The term  $\mathbb{E}_{\mathcal{D}}(g^{(\mathcal{D})}(x))$  gives an **average function** that we denote by  $\tilde{g}(x)$

$$\begin{aligned}\mathbb{E}_{\mathcal{D}}[E_{out}(g^{(\mathcal{D})})] &= \mathbb{E}_x[\mathbb{E}_{\mathcal{D}}(g^{\mathcal{D}}(x)^2) - 2 \tilde{g}(x) f(x) + f(x)^2] \\ &= \mathbb{E}_x\left[\underbrace{\mathbb{E}_{\mathcal{D}}(g^{\mathcal{D}}(x)^2) - \tilde{g}(x)^2}_{\mathbb{E}_{\mathcal{D}}[(g^{(\mathcal{D})}(x) - \tilde{g}(x))^2]} + \underbrace{\tilde{g}(x)^2 - 2 \tilde{g}(x) f(x) + f(x)^2}_{(\tilde{g}(x) - f(x))^2}\right] \\ &\quad \text{variance}(x) \qquad \qquad \qquad \text{bias}(x)\end{aligned}$$

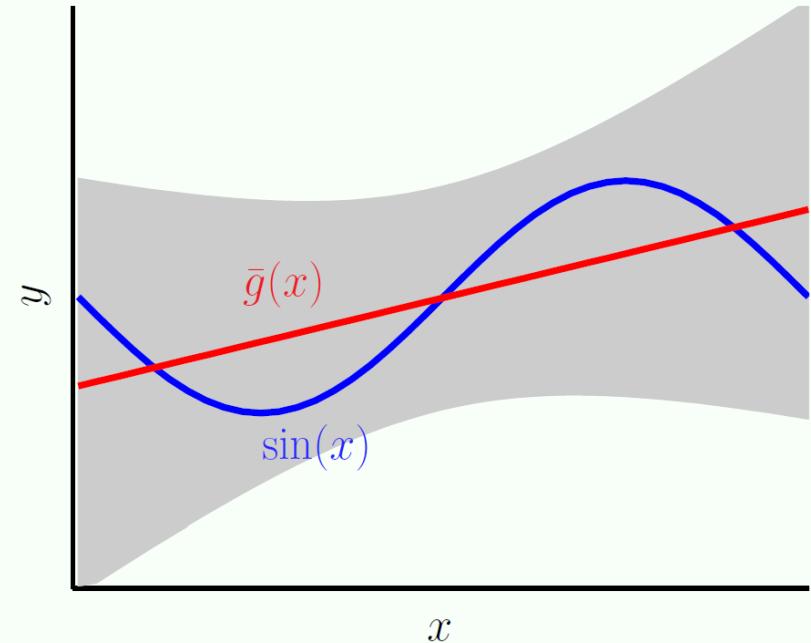
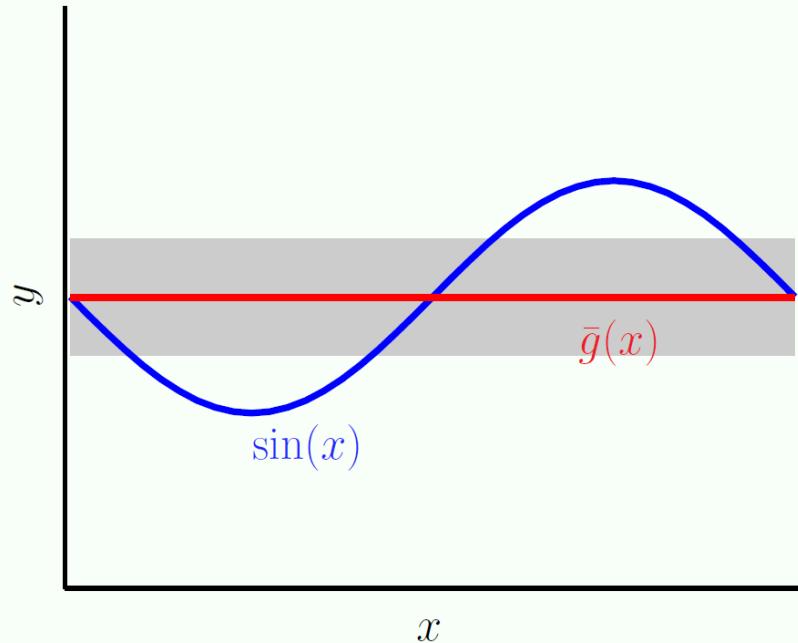
$$\mathbb{E}_{\mathcal{D}}[E_{out}(g^{(\mathcal{D})})] = \mathbb{E}_x[\text{bias}(x) + \text{variance}(x)] = \text{bias} + \text{variance}$$

# Bias-Variance Tradeoff: Comments



- $\mathbb{E}_{\mathcal{D}}[E_{out}(g^{(\mathcal{D})})] = \sigma^2 + \text{bias} + \text{variance}$  ( for noisy signals)
  - $\sigma^2$  is the variance of the noise
  - The noise is unavoidable no matter what we do, so our interest remains in bias and variance
  - Unfortunately it is impossible to compute bias and variance. Thus, the bias-variance decomposition is a conceptual tool which is helpful when it comes to developing a model.
- There are two typical goals when we consider bias and variance:
  - To lower the variance without significantly increase the bias (1)
  - To lower the bias without significantly increase the variance (2)
- These goals are achieved by different techniques: Regularization(1) , prior knowledge (2)

# Back to $\mathcal{H}_0$ and $\mathcal{H}_1$ ; and, our winner is . . .

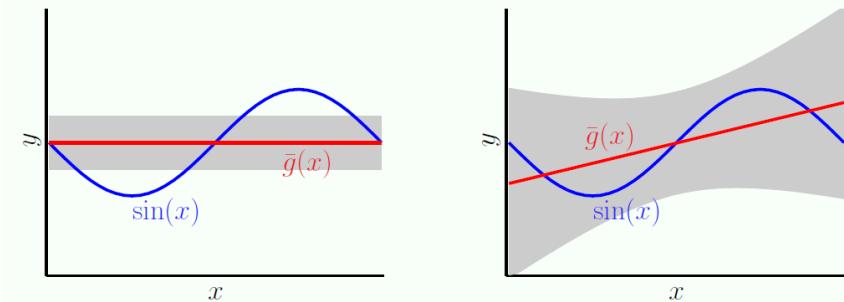
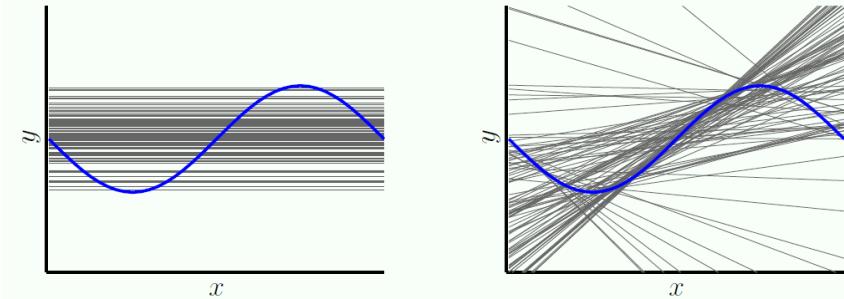


$$\begin{aligned} \mathcal{H}_0 \\ \text{bias} &= 0.50 \\ \text{var} &= 0.25 \\ \hline E_{\text{out}} &= 0.75 \quad \checkmark \end{aligned}$$

$$\begin{aligned} \mathcal{H}_1 \\ \text{bias} &= 0.21 \\ \text{var} &= 1.69 \\ \hline E_{\text{out}} &= 1.90 \end{aligned}$$

# Match Learning Power to Data, . . . Not to $f$

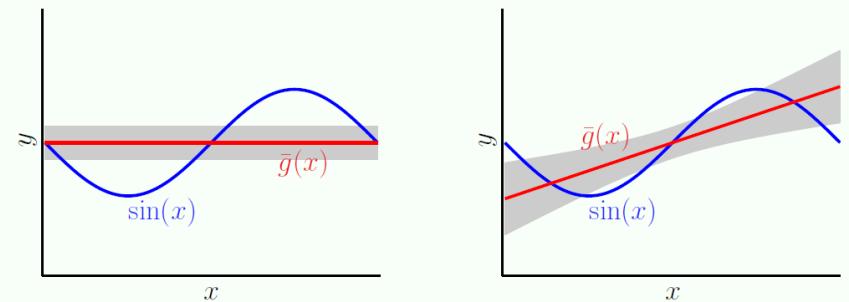
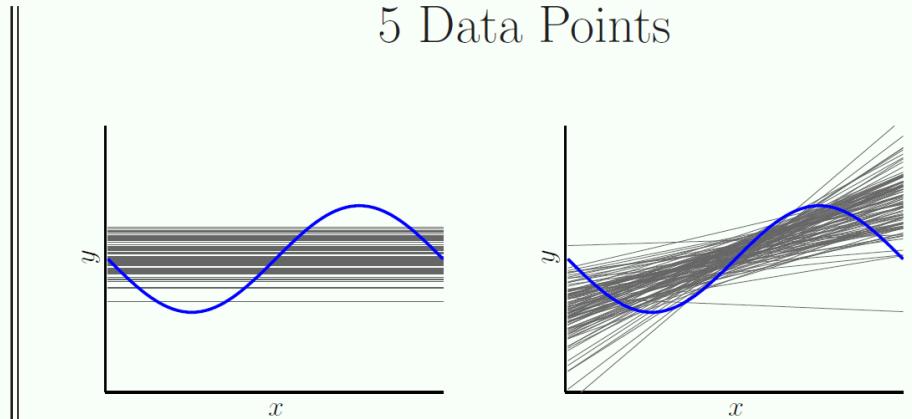
2 Data Points



$$\begin{aligned} \mathcal{H}_0 \\ \text{bias} &= 0.50; \\ \text{var} &= 0.25. \\ \hline E_{\text{out}} &= 0.75 \quad \checkmark \end{aligned}$$

$$\begin{aligned} \mathcal{H}_1 \\ \text{bias} &= 0.21; \\ \text{var} &= 1.69. \\ \hline E_{\text{out}} &= 1.90 \end{aligned}$$

5 Data Points

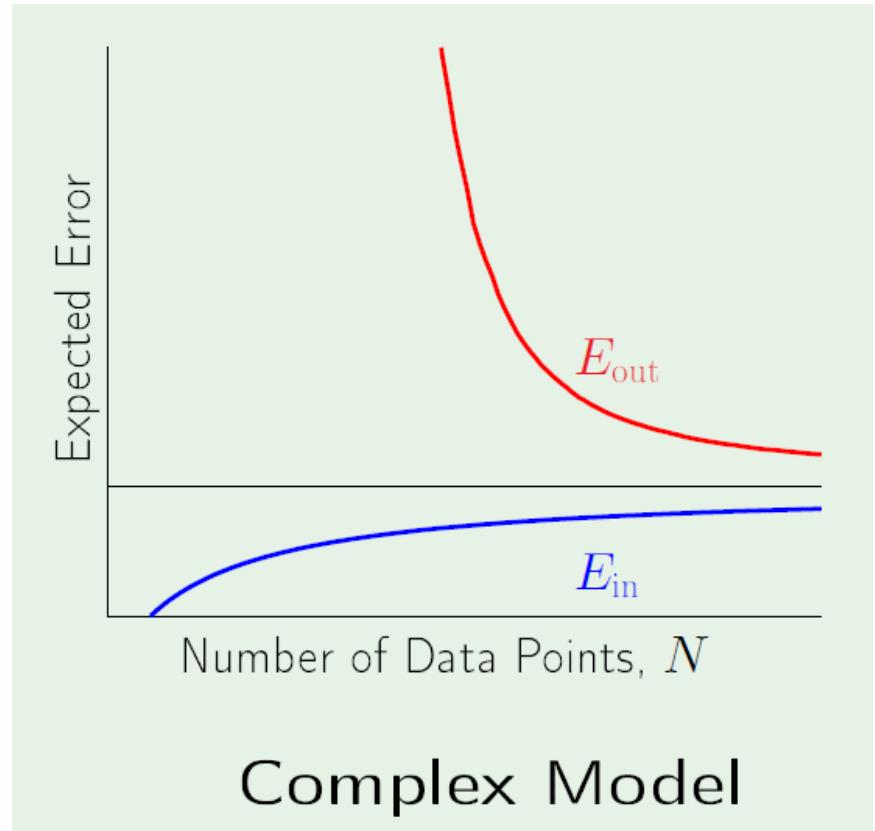
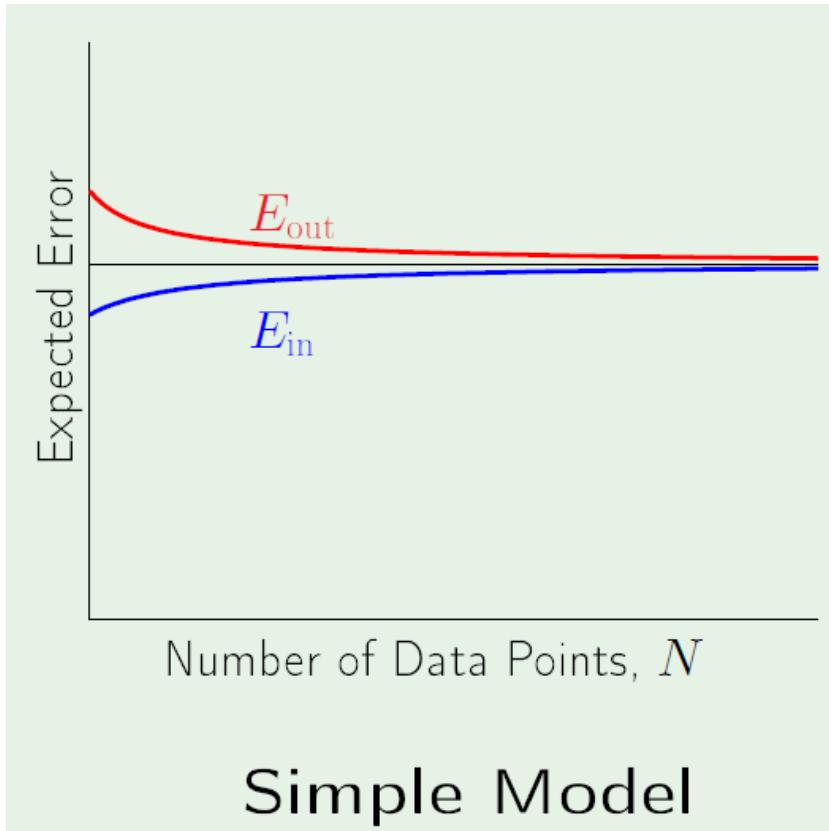


$$\begin{aligned} \mathcal{H}_0 \\ \text{bias} &= 0.50; \\ \text{var} &= 0.1. \\ \hline E_{\text{out}} &= 0.6 \end{aligned}$$

$$\begin{aligned} \mathcal{H}_1 \\ \text{bias} &= 0.21; \\ \text{var} &= 0.21. \\ \hline E_{\text{out}} &= 0.42 \quad \checkmark \end{aligned}$$

# Learning Curve

- The learning curves summarize the behaviour of the errors  $\mathbb{E}_{\mathcal{D}}[E_{in}(g^{(\mathcal{D})})]$  and  $\mathbb{E}_{\mathcal{D}}[E_{out}(g^{(\mathcal{D})})]$  when we vary the size N of the training set .



The model complexity influence the Expected Error and the speed of convergence

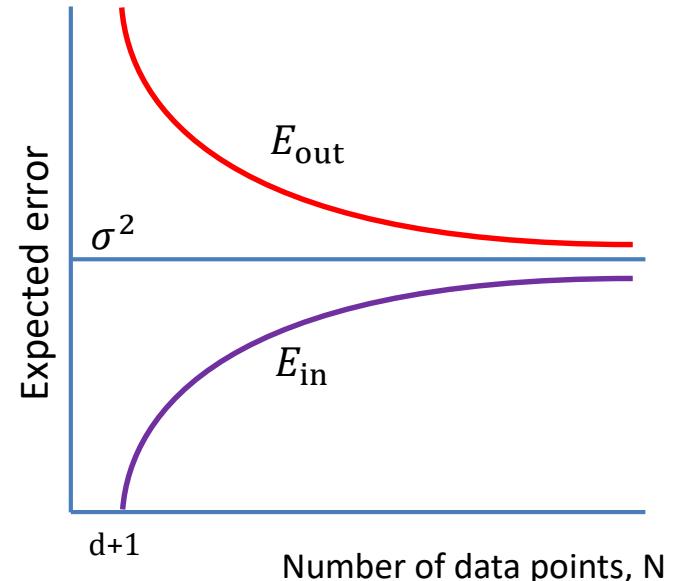
# Learning Curve for Linear Regression

- Let now consider the expression for the expected values of  $E_{\text{in}}(\mathbf{w}_{\text{lin}})$  and  $E_{\text{out}}(\mathbf{w}_{\text{lin}})$

$$\mathbb{E}_{\mathcal{D}}[E_{\text{in}}(\mathbf{w}_{\text{lin}})] = \sigma^2 \left(1 - \frac{d+1}{N}\right), \text{ for } N \geq d+1$$

$$\mathbb{E}_{\mathcal{D}}[E_{\text{test}}(\mathbf{w}_{\text{lin}})] = \sigma^2 \left(1 + \frac{d+1}{N}\right) \quad (\text{approx. to } E_{\text{out}})$$

The figure shows the linear regression learning curve under the OLS assumptions.



- $E_{\text{in}}$  : When  $N$  increase the model absorbs as much information as possible with  $d+1$  parameters
- $E_{\text{out}}$  : When  $N$  increase the out of sample error of the model decreases to the residual noise.
- This behaviour of the learning curve is the expected when the right complexity model has been chosen

# EVALUATION METRICS

# Confusion Matrix-Based Performance Measures

True class → Hypothesized   class V	Pos	Neg
Yes	TP	FP
No	FN	TN
	P=TP+FN	N=FP+TN

A Confusion Matrix

- **Multi-Class Focus:**
  - **Accuracy** =  $(TP+TN)/(P+N)$
- **Single-Class Focus:**
  - **Precision** =  $TP/(TP+FP)$
  - **Recall** =  $TP/P$
  - **Fallout** =  $FP/N$
  - **Sensitivity** =  $TP/(TP+FN)$
  - **Specificity** =  $TN/(FP+TN)$

## Aliases and other measures

- Accuracy = 1 (or 100%) - Error rate
- Recall = TPR = Hit rate = Sensitivity
- Fallout = FPR = False Alarm rate
- Precision = Positive Predictive Value (PPV)
- Negative Predictive Value (NPV) =  $TN/(TN+FN)$
- Likelihood Ratios:
  - $LR+ = \text{Sensitivity}/(1-\text{Specificity})$
  - $LR- = (1-\text{Sensitivity})/\text{Specificity}$

# Pairs of Measures and Compounded Measures

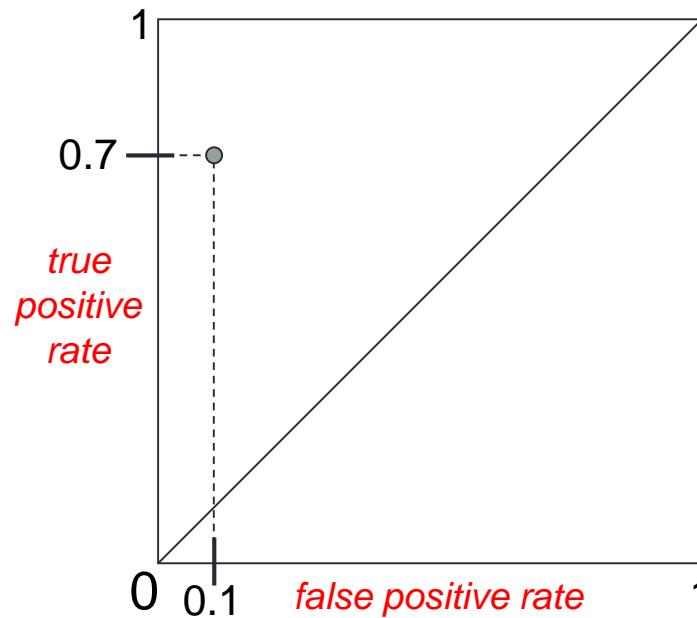
- Precision / Recall
- Sensitivity / Specificity
- Likelihood Ratios (LR+ and LR-)
- Positive / Negative Predictive Values
- F-Measure:
  - $F\alpha = [(1+\alpha) (\text{Precision} \times \text{Recall})] / [(\alpha \times \text{Precision}) + \text{Recall}]$        $\alpha = 1, 2, 0.5$
- G-Mean:      2-class version      single-class version
  - $\text{G-Mean} = \text{Sqrt}(TPR \times TNR)$  or  $\text{Sqrt}(TPR \times \text{Precision})$

# Evaluating the output

How can we measure the performance of a deterministic output?

“recall”

$$\frac{\# \text{ true positives (TP)}}{\# \text{ positives (TP+FP)}}$$



*true  
positive  
rate*

$$\frac{\# \text{ false positives (FP)}}{\# \text{ negatives ( TN+FP)}} \\ 1 - \text{“specificity”}$$

# ROC Curve

How can we measure the performance of a probability classifier?

“recall”

$$\frac{\# \text{ true positives (TP)}}{\# \text{ positives (TP+FP)}}$$

AUC: Area under the curve

the AUC of a classifier is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance

Variance estimation (ROC Curve) is needed to compare two classifiers

