$\mathcal{H}$-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/features).
- This is, $h(\boldsymbol{x}) = w_0 + w_1 x_1 + \cdots + w_d x_d = w_0 + \sum_{i=1}^{d} x_i w_i$
  - Equivalently using $\boldsymbol{x}^T = (1, x_1, x_2, \ldots, x_d)$, $\mathbf{w}^T = (w_0, w_1, w_2, \ldots, w_d)$
  - We can write: $h(\boldsymbol{x}) = \mathbf{w}^T \boldsymbol{x}$
- Three different tasks 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: $h(\mathbf{x}) = \sum_{i=0}^{d} w_i \, x_i = \mathbf{w}^{\mathsf{T}}\mathbf{x}$

Elements of the problem

The data set: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)$

y_n is the credit line for customer $\mathbf{x}_n$ (feature vector)

We've asumed $h_w(\mathbf{x}) = w_0 + w_1 x_1 + \cdots + w_d x_d = \mathbf{w}^{\mathrm{T}}\mathbf{x}$

For each given $\mathbf{w}$ we compute the sample values $h_w(x_1), \ldots, h_w(x_n)$

To measure the Error / Loss of h_w with respect to f, a quadratic error is used in regression.

$$\mathbf{Error}(\mathbf{x}) = (\boldsymbol{h_w}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{x}))^2$$

Then the sample error is

$$E_{in}(h_w) = \frac{1}{N}\sum_{n=1}^{N}(h_w(x_n) - y_n)^2$$

How to carry on with this ?

Linear Regression

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

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

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

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

- In order to find $\widehat{h}$ we minimize the Empirical Risk (ERM):

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

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

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

When is the quadratic error optimal?

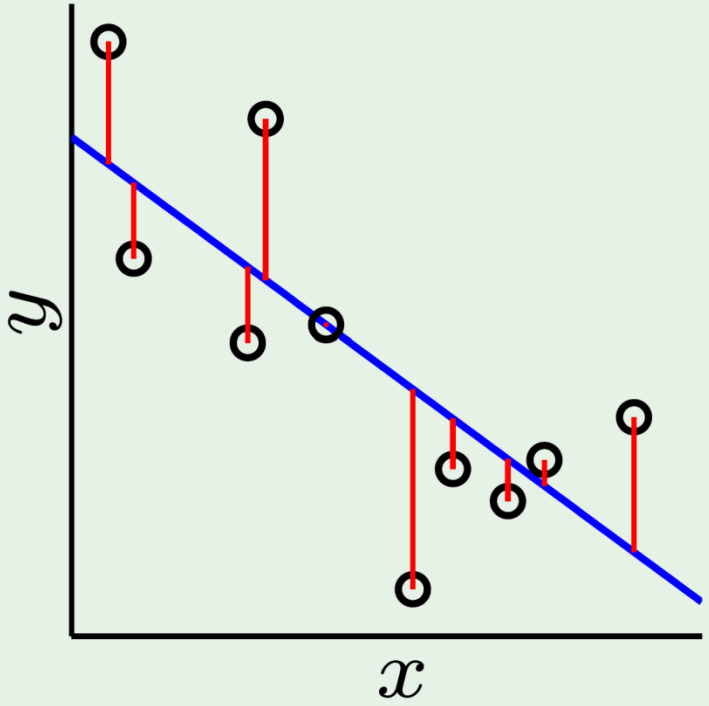
- The Ordinary Least Square model only assumes error in the dependent variable
  - This hypothesis is not valid in all cases
  - But is a good approximation in many cases

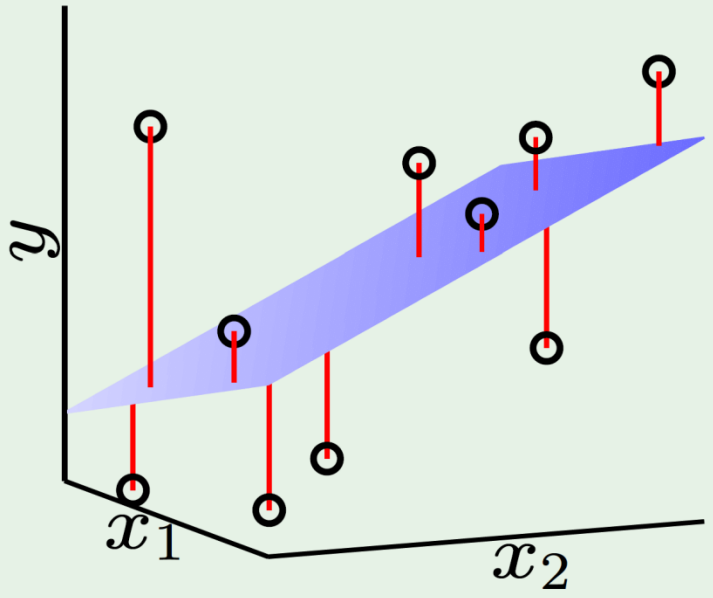
- Model: $y_i = f(\boldsymbol{x}_i, \boldsymbol{\beta}) + noise$, f linear in β

- **Theorem of Gauss–Markov:** Under the assumption of uncorrelated noise of zero mean and bounded variance, the OLS technique reachs the minimum variance unbiased estimator for $\boldsymbol{\beta}$

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

Illustration of Linear Regression





A matrix expression for E_{in}

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}\left(\mathbf{w}^{\mathsf{T}}\mathbf{x}_n - y_n\right)^2$$

$$= \frac{1}{N}\|\mathrm{X}\mathbf{w} - \mathbf{y}\|^2$$

where $\mathrm{X} = \begin{bmatrix} —\mathbf{x}_1{}^{\mathsf{T}}— \\ —\mathbf{x}_2{}^{\mathsf{T}}— \\ \vdots \\ —\mathbf{x}_N{}^{\mathsf{T}}— \end{bmatrix}$, $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$

Minimizing E_{in} : normal equations

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

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

$$\mathrm{X}^{\mathsf{T}}\mathrm{X}\mathbf{w} = \mathrm{X}^{\mathsf{T}}\mathbf{y}$$

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

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

Math details

- To compute $\frac{\partial E_{in}}{\partial \boldsymbol{w}}$ we make use of two well known properties:
  - $\frac{\partial}{\partial \boldsymbol{x}}(\boldsymbol{x}^T A \boldsymbol{x}) = (A^T + A)\boldsymbol{x}, \quad \frac{\partial}{\partial \boldsymbol{x}}(\boldsymbol{x}^T A) = A$

- $$\frac{\partial E_{in}}{\partial \mathbf{w}} = 2X^T X\mathbf{w} - 2X^T \boldsymbol{y} = \mathbf{0} \rightarrow \quad \mathbf{w}_{\text{lin}} = (X^T X)^{-1} X^T y$$

- Can we always compute $(X^T X)^{-1}$?
  - Let consider the **Singular Value Decomposition (SVD) : X=UDV^T**
  - $X^T X = V\,DDV^T$
  - $(X^T X)^{-\mathbf{1}} = V^{-T} D^{\dagger} V^{-1} = V D^{\dagger} V^T$ this product can always be computed
  - When the range($X^T X$) = d, there exist only one solution.
  - When the range($X^T X$) < d, there exist infinite solutions

Some vector notation

- $\|A\|_F = \sqrt{\sum_{ij} a_{ij}^2}$
- $\sum_{ij}\left(x_{ij}w_j - y_i\right)^2 = \sum_i\left(\boldsymbol{x}_i^T\boldsymbol{w} - y_i\right)^2 = \|X\boldsymbol{w} - \boldsymbol{y}\|^2$
- $\|X\boldsymbol{w} - \boldsymbol{y}\|^2 = (X\boldsymbol{w} - \boldsymbol{y})^T(X\boldsymbol{w} - \boldsymbol{y})$

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

$$= \boldsymbol{w}^TX^TX\boldsymbol{w} - 2\boldsymbol{y}^TX\boldsymbol{w} + \boldsymbol{y}^T\boldsymbol{y}$$

A Linear Regression Algorithm

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

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

2: Compute the pseudo-inverse $\mathrm{X}^\dagger = (\mathrm{X}^\mathsf{T}\mathrm{X})^{-1}\mathrm{X}^\mathsf{T}$.

3: Return $\mathbf{w} = \mathrm{X}^\dagger \mathbf{y}$.

Linear Regression properties

- **HAT-MATRIX (H):** Let X (N×(d+1)) denote the samples

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

- The $\mathbf{w}_{lin}$ is an attempts to map the input X to the output $\hat{\boldsymbol{y}}$, *but*

$$\hat{\mathbf{H}} = \mathrm{X}\,(X^T X)^{-1} X^T \textit{ map } \boldsymbol{y} \textit{ to } \hat{\boldsymbol{y}}$$

- Projection matrix $\hat{\mathbf{H}}$:
  - The properties of $\hat{\mathbf{H}}$ are relevant in the analysis of E_{out} and E_{in}
  - The hat matrix is an **idempotent** projection matrix**,** this means $\hat{\mathbf{H}}^2 = \hat{\mathbf{H}}$
  - The trace($\hat{\mathbf{H}}$) =d+1 (input vector dimensión +1)

- **Generalization Error:** For linear regression exact formulas for the expected E_{out} and E_{in} can be found

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

- (see exercises in the book)

Alternative: Gradient Descent

- Given $\boldsymbol{w}_0$ we want to find $\hat{\mathbf{v}}$ such that $E_{in}\left(\boldsymbol{w}_0 + \eta\widehat{\mathbf{v}}\right) < E_{in}(\boldsymbol{w}_0)$
- How decrement $E_{in}(\boldsymbol{w}_0)$? : Apply Taylor expansion to first order, whith $\|\hat{\mathbf{v}}\| = 1$

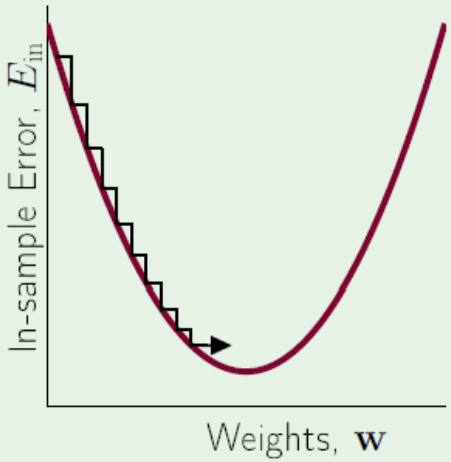
$$\Delta E_{in} = E_{in}(\boldsymbol{w}(0) + \eta\hat{\mathbf{v}}) - E_{in}\left(\boldsymbol{w}(0)\right)$$
$$= \eta\nabla E_{in}\left(\boldsymbol{w}(0)\right)^{\mathrm{T}}\hat{\mathbf{v}} + \mathcal{O}(\eta^2)$$
$$\geq -\eta\left\|\nabla E_{in}\left(\boldsymbol{w}(0)\right)\right\|$$

The equality holds if y only if $\hat{\mathbf{v}} = -\frac{\nabla E_{in}\left(\boldsymbol{w}(0)\right)}{\|\nabla E_{in}(\boldsymbol{w}(0))\|}$ (negative gradient)

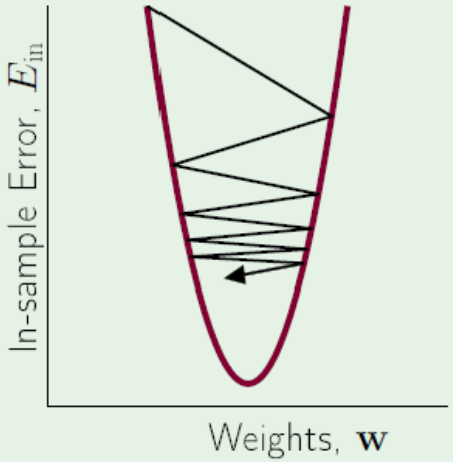
- Gradient Descent (GD):
  - Gradient descent is a general iterative optimization technique that reach a local optimum following the direction of the gradient vector on each point.

How to fix η ?

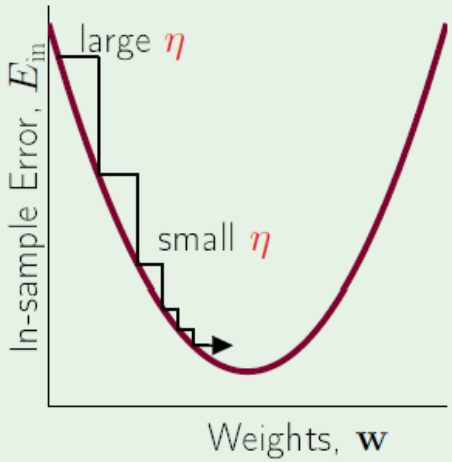
How η affects the algorithm:



η too small



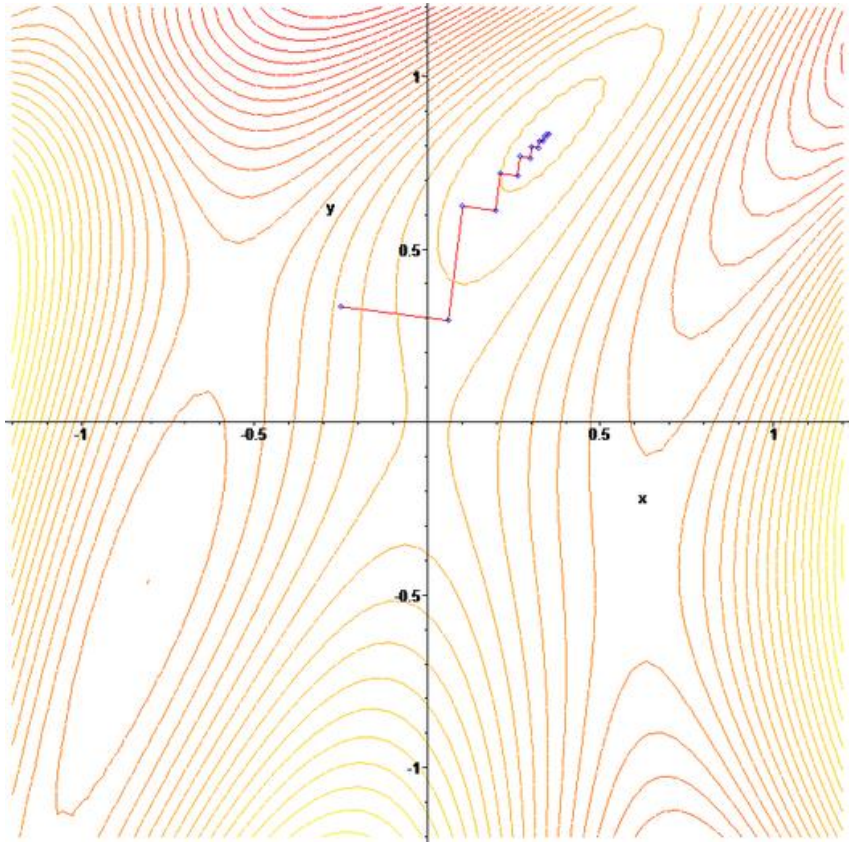
η too large

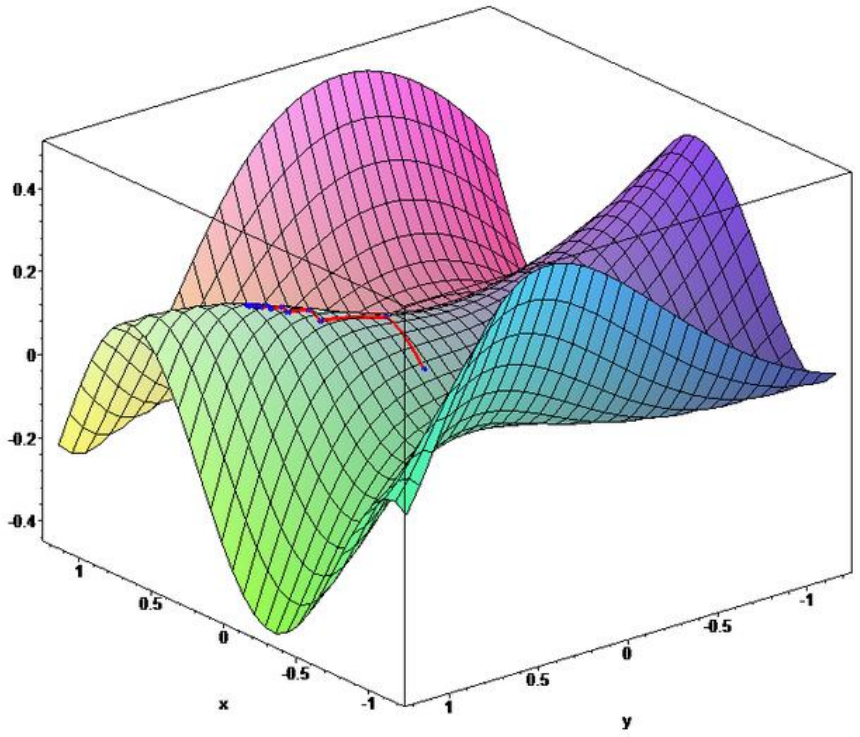


variable η – just right

η should increase with the slope

What optimum do we achieve ?





- https://en.wikipedia.org/wiki/Gradient_descent

Gradient Descent in Regression

It starts on some initial value $\mathbf{w_0}$ 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, . . . , n).Here, η is called the learning rate.

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

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

It has been found the use of all points precludes reaching good optima.

Stochastic Gradient Descent: SGD

- An alternative is to use a stochastic estimation by selecting a part of the sample to compute the gradient, M < < 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 (a rule of thumb is size: 32-256)

An iterative regression algorithm

Batch Gradient Descend

- Given the data set $(\mathbf{x}_n, y_n), n = 1,2,\cdots,N$

1. Fix **w**=0 , $\eta = \eta_0$
2. Iterate
   For j=0,..,K :
   $w_j := w_j - \eta \sum_{n=1}^{N} x_{nj}(\mathbf{h}(\mathbf{x_n}) - y_n)$ (**all sample participate**)
3. Until $E_{in}(\mathbf{w})$ < epsilon

Stochastic Gradient Descend

1. Fix **w**=0 , $\eta = \eta_0$
2. Iterate:
3. **Shuffle and Split the sample into a sequence of mini-batches**
4. Iterate on mini-batches
   For j=0,..,K :
   $w_j := w_j - \eta \sum_{n \epsilon Minibatch} x_{nj}(\mathbf{h}(\mathbf{x_n}) - y_n)$ (**only a mini-bacth participate**)
5. Until $E_{in}(\mathbf{w})$ < epsilon

Newton's Method: a cure for oscillation

- A new update rule for **w** based on the second order derivatives (Hessian)

$$f(x+\Delta x, y+\Delta y) \approx f(x,y) + \frac{\partial f}{\partial x}\Delta x + \frac{\partial f}{\partial y}\Delta y + \frac{1}{2}\left(\frac{\partial^2 f}{\partial x^2}\Delta x^2 + \frac{\partial^2 f}{\partial y^2}\Delta y^2 + \frac{\partial^2 f}{\partial y \partial x}\Delta x \Delta y + \frac{\partial^2 f}{\partial x \partial y}\Delta x \Delta y\right)$$

- Let's approach $E_{\text{in}}(\mathbf{w})$ up to second order.

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

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

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

Now the matrix **H** defines the advance on the gradient direction

The major complexity is to invert **H**

Summary of Linear Regression

- Now $\mathcal{X} = \{1\} \times \mathbb{R}^d,\ \mathcal{Y} = \mathbb{R} \quad and \quad f: \mathcal{X} \to \mathcal{Y}$
- We assume i.i.d. samples from $\mathrm{P}(\mathcal{X} \times \mathcal{Y})$
- Functions in $\mathcal{H}$ take the form: $h(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}$
- Learning criteria (ERM): $\mathbf{w}^* = argmin_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^{N} error_h(\boldsymbol{x_i}, y_i)$
- Quadratic loss/error function: $loss_h(\boldsymbol{x_i}, y_i) = (h(\boldsymbol{x}_i) - y_i)^2 = (\mathbf{w}^T \boldsymbol{x}_i - y_i)^2$
- Two learning algorithms:
  - Stochastic Gradient descend (SGD) / Gradient descend (GD)
  - Pseudo inverse matrix

In practice

- Analyzing the data:
  - Regression 1D allows residues analysis to detect samples out of the OLS hypothesis

- Analyzing the residuals with graphs (small samples):
  - Correlated errors
  - Errors with non-constant variance
  - Dependence between predictors
  - Outliers: (relevant in small samples)

- In higher dimensions or very large data samples, the scenario is more complex and the original hypothesis loses relevance.

- Categorical predictors
- Lost values

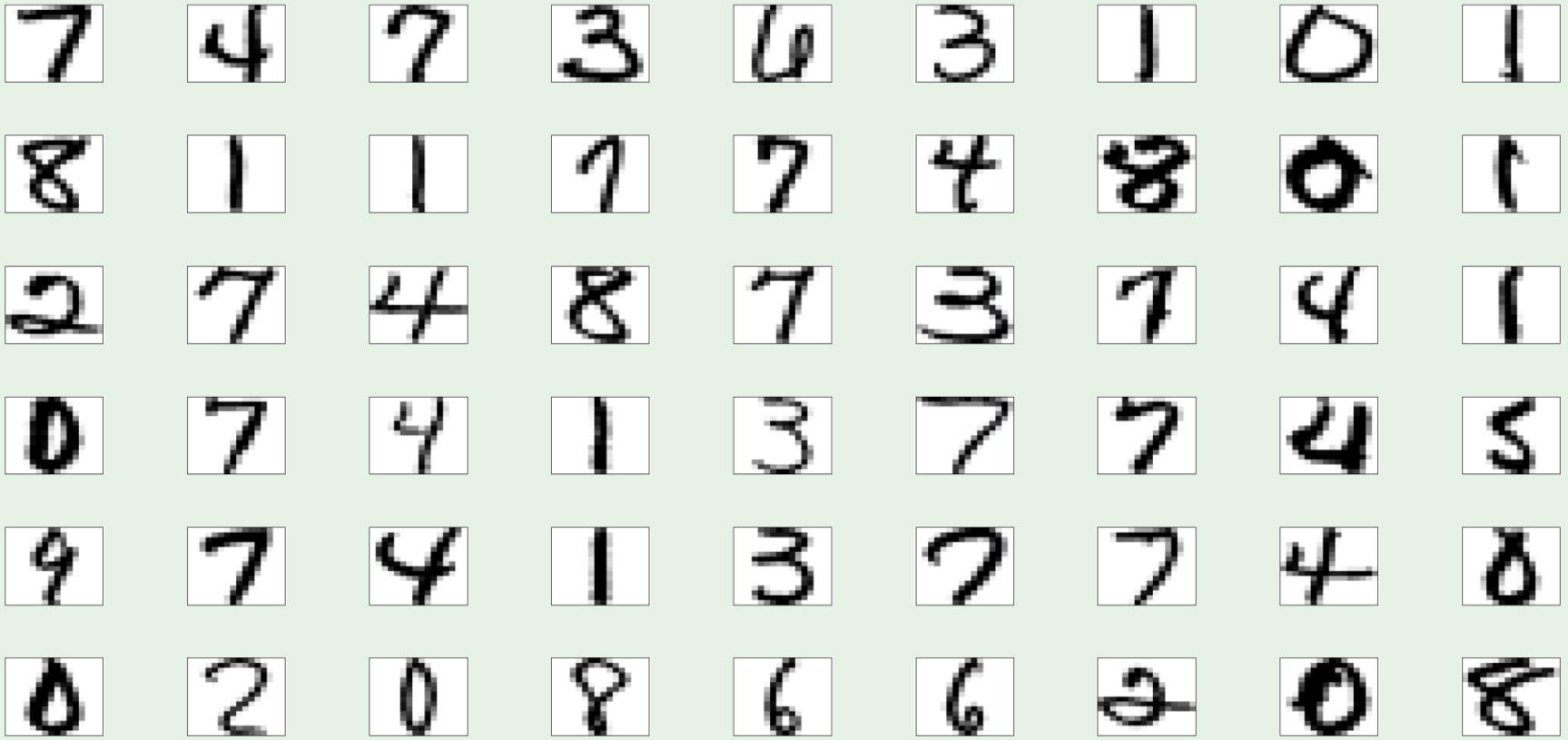
Categorical features & Lost values

- Categorical features encoding :
  - For lineal models a categorical feature is encoded as an one-hot vector: [0,0,0,..,1,..,0,0] wich length is the number categorical values
  - Some non-lineal models can manage categorical features: Trees
- Lost values:
  - Linear models can not manage lost values
  - Surrogate values:
    - Each value can be interpolated from its feature histogram
    - Each value can be predicted using a specific model
  - Some non-lineal models can manage this situation: trees

Classification

Data Set

A real data set



Input Representation

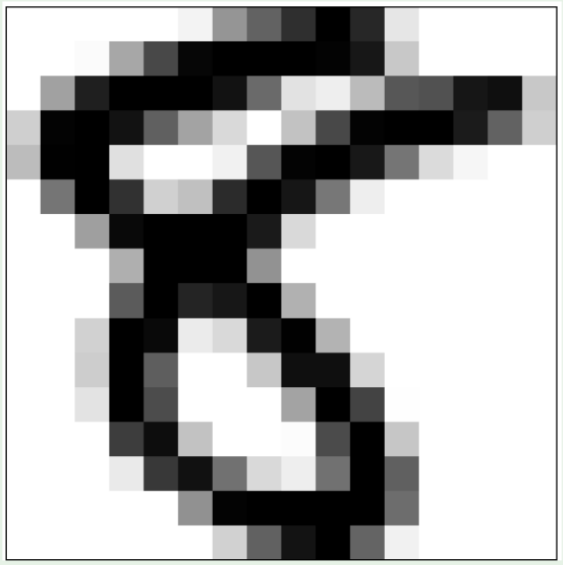
'raw' input $\mathbf{x} = (x_0, x_1, x_2, \cdots, x_{256})$

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

Features: Extract useful information, e.g.,

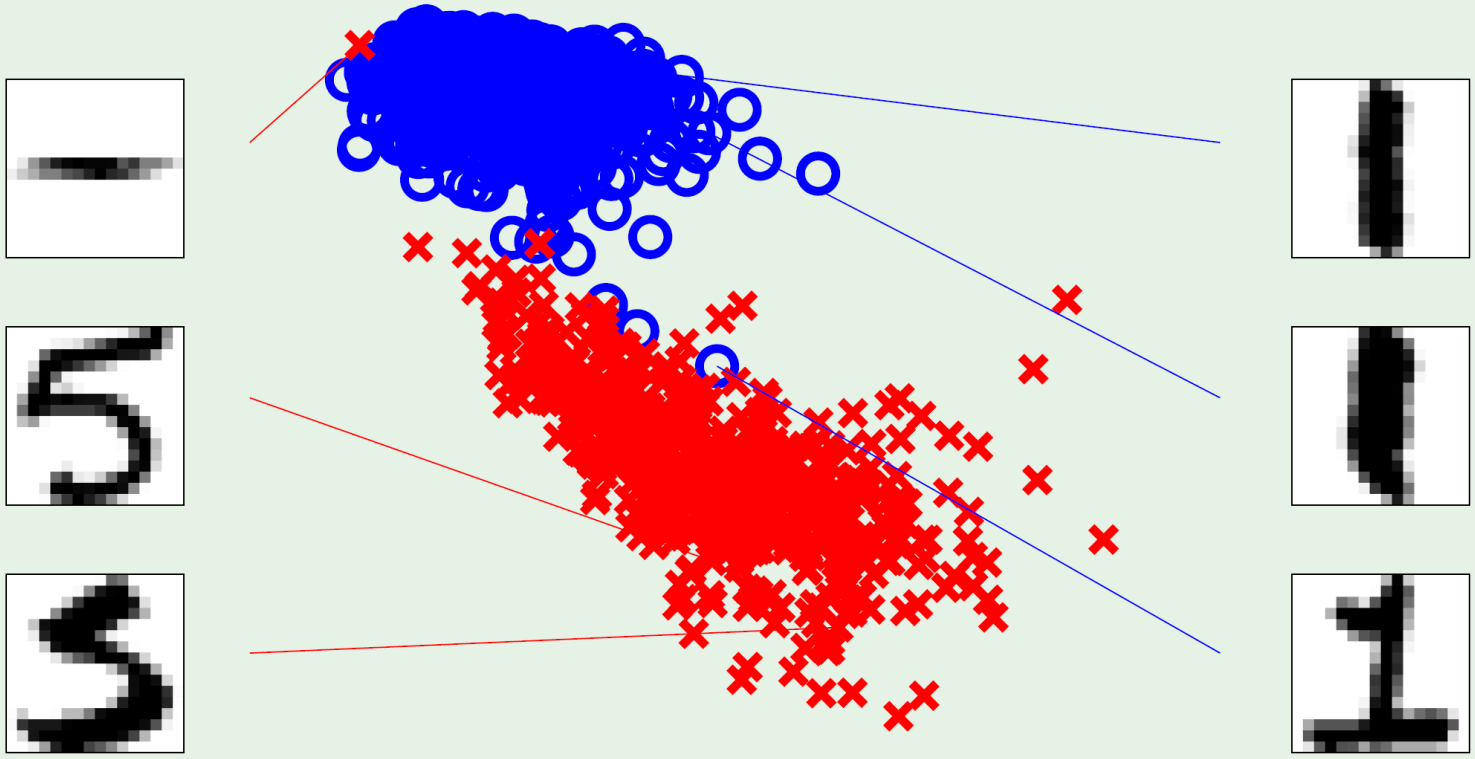
intensity and symmetry $\mathbf{x} = (x_0, x_1, x_2)$

linear model: (w_0, w_1, w_2)

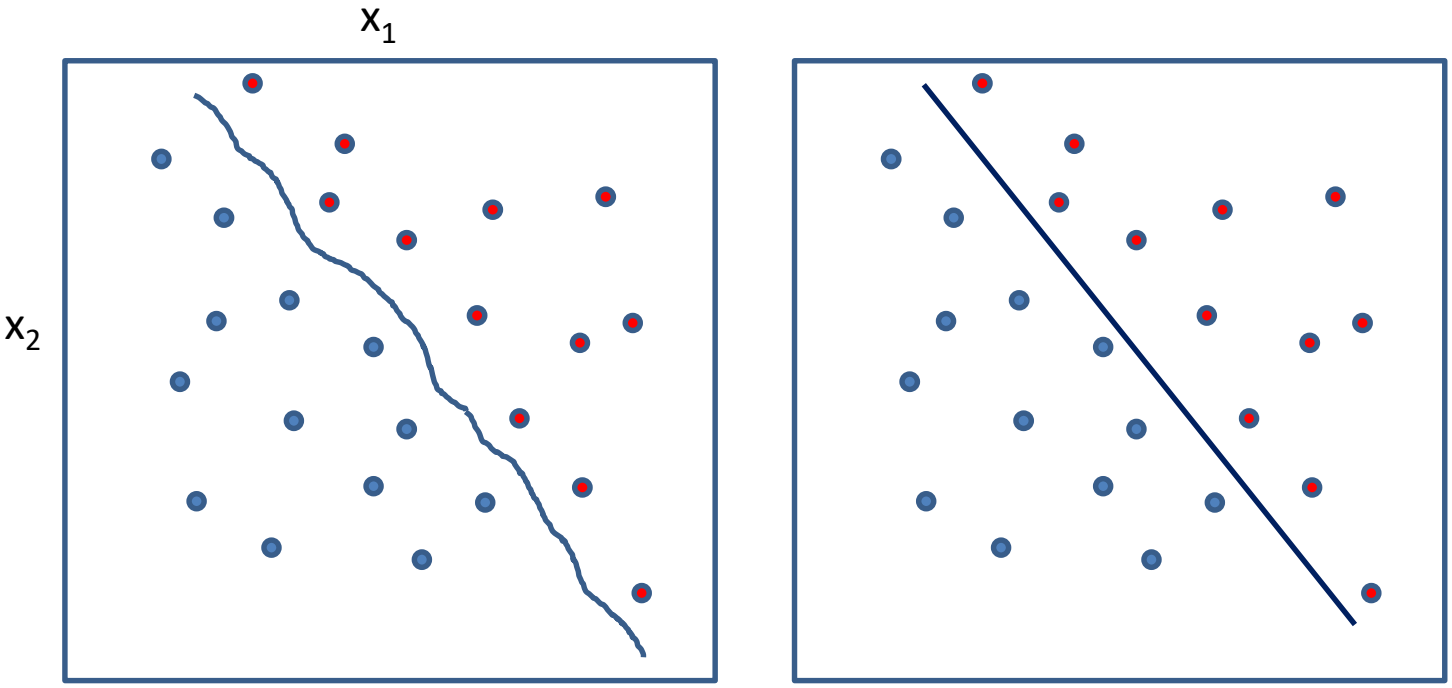


Ilustration of features

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



A very simple case



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

Perceptron- Two classes

- Assigment Rule (CREDIT PROBLEM):

  if $\sum_{i=1}^{d} x_i w_i \geq b$ $(Threshold)$ THEN Approve credit

  if $\sum_{i=1}^{d} x_i w_i < b$ $(Threshold)$ THEN Denay credit

- Then, $h(\boldsymbol{x}) = sign\left(\left(\sum_{i=1}^{d} x_i w_i\right) + b\right)$, $h(x) \in \{-1,1\}$

  - Equivalently using $\boldsymbol{x}^T = (1, x_1, x_2, \ldots, x_d)$, $\mathbf{w}^T = (b, w_1, w_2, \ldots, w_d)$
  - We can write:

$$\mathcal{H} = \{h |\, h(\boldsymbol{x}) = \text{sign}(\mathbf{w}^T \boldsymbol{x}) \text{ , for } \mathbf{w} \text{ fixed } \}$$

Perceptron or lineal separator

This new class $\mathcal{H}$ is called *Perceptron or lineal separator*

Learning Criteria (ERM) : $\underbrace{\min_{w \in R^{d+1}} \frac{1}{N} \sum_{n=1}^{N} [\![\text{sign}(\mathbf{w}^T \mathbf{x}_n) \neq y_n]\!]}_{E_{in}}$

How to minimize it ?

Now a new Error/Loss function appear: $[\![\text{sign}(\mathbf{w}^T \mathbf{x}_n) \neq y_n]\!]$

ERM is a non-derivable discrete function….. different from regression

Let's analyze in more detail

- The error function is

$$error(\mathbf{w}^T\mathbf{x}_n, y_n) = [\![\text{sign}(\mathbf{w}^T\mathbf{x}_n) \neq y_n]\!] = \begin{cases} 1 \; if \; \text{sign}(\mathbf{w}^T\mathbf{x}_n) \neq y_n \\ 0 \; if \; \text{sign}(\mathbf{w}^T\mathbf{x}_n) = y_n \end{cases}$$

- THEN a new and derivable error can be deduced from it

$$error(\mathbf{w}^T\mathbf{x}_n, y_n) = \begin{cases} -y_n\mathbf{w}^T\mathbf{x}_n \; if \; \text{sign}(\mathbf{w}^T\mathbf{x}_n) \neq y_n \\ 0 \qquad\qquad\quad if \; \text{sign}(\mathbf{w}^T\mathbf{x}_n) = y_n \end{cases}$$

- Equivalently,

$$error(\mathbf{w}^T\mathbf{x}_n, y_n) = \max(0, -y_n\mathbf{w}^T\mathbf{x}_n)$$

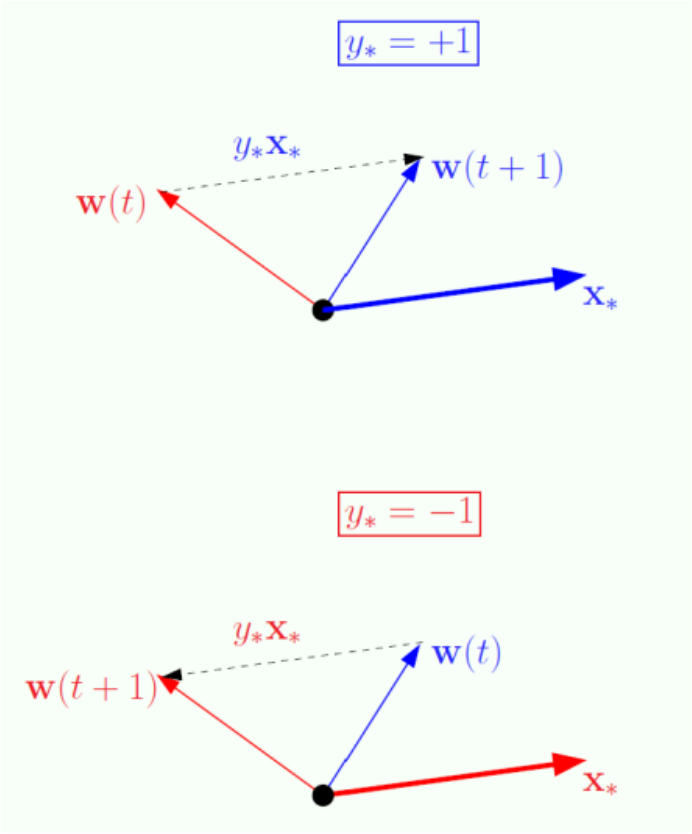
Perceptron Learning Algoritm (PLA)

Stochastic Gradient Descend with batch size =1 and learning rate λ=1

ADAPTING RULE :

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

- **The learning algorithm iterates on the data sample but memoryless**
- Simple algorithm searching for the best, $h \in \mathcal{H}$, through iterative evaluations of the training data ($\mathcal{D}$):
- **For** linearly separable data set:
  - The PLA algorithm is always able to find a vector ŵ such that $h(\boldsymbol{x}_i) = y_i$ 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 $(\mathbf{x}_n, y_n), n = 1,2,\cdots,N$
  - Step.1: Fix $\mathbf{w}_{\text{ini}}=0$
  - Step.2: Iterate on the $\mathcal{D}$-samples improving the solution:
  - repeat

    For each $\boldsymbol{x}_i \in \mathcal{D}$ **do**

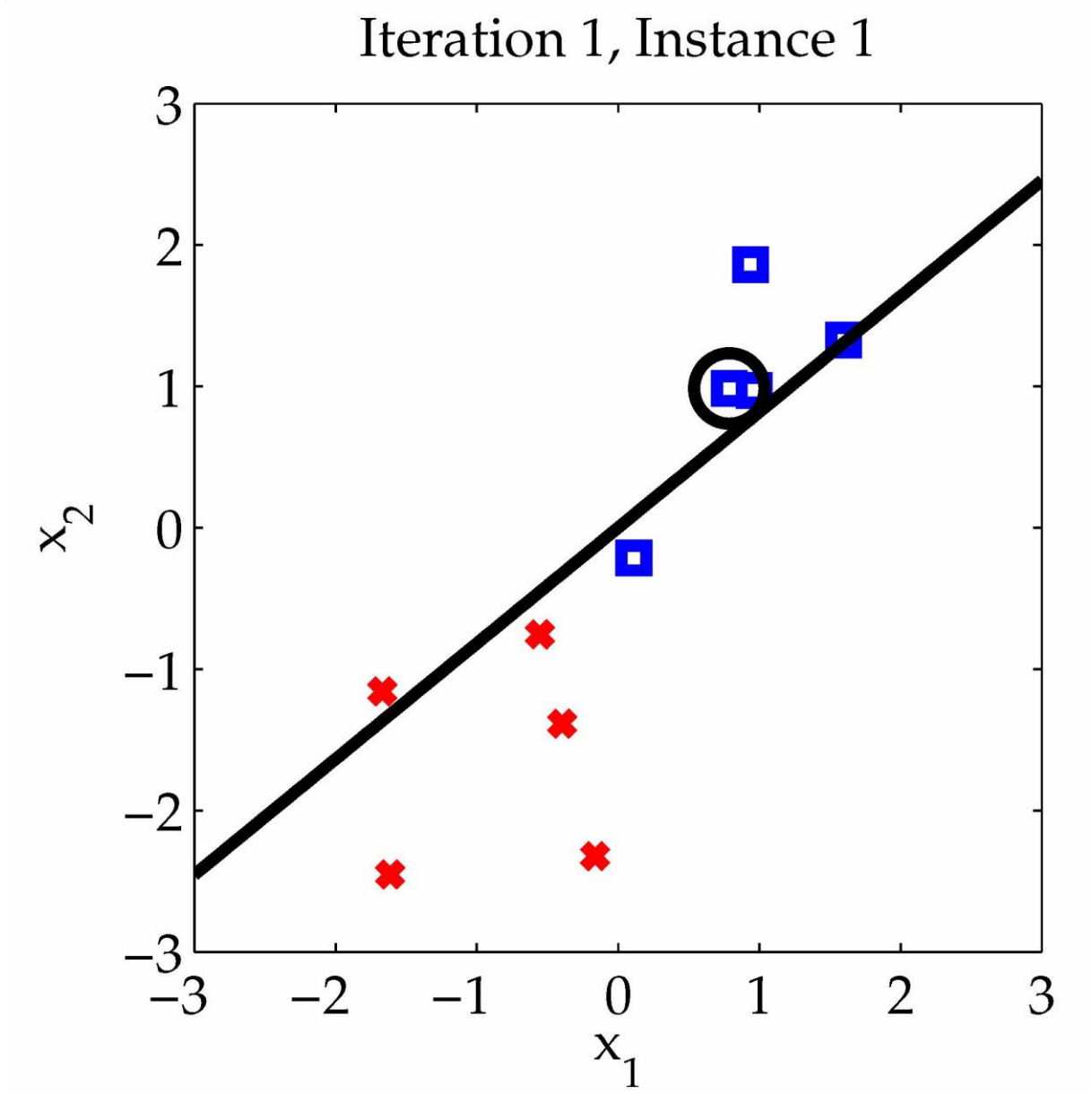
    if: $sign(\boldsymbol{w}^T \boldsymbol{x}_i) \neq y_i$ **then**

    update $\mathbf{w}$: $\mathbf{w}_{\text{new}} = \mathbf{w}_{\text{old}} + y_i \mathbf{x}_i$

    else continue

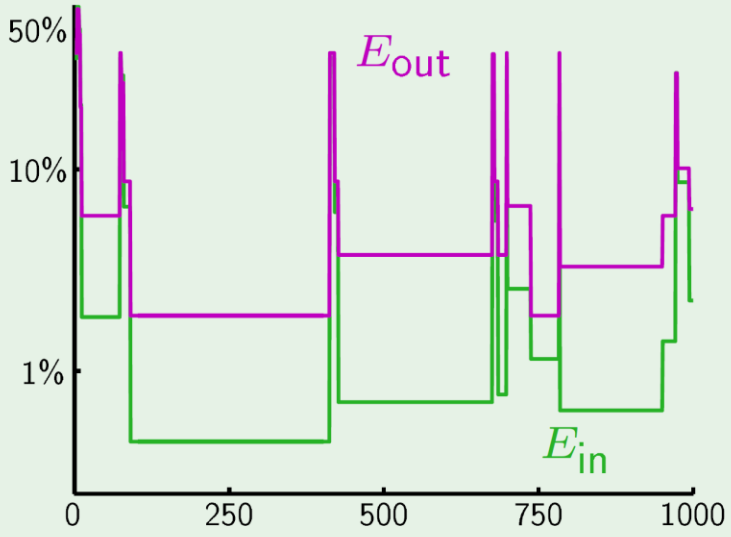
    End for
  - Until No changes in a full pass on $\mathcal{D}$

How the perceptron works

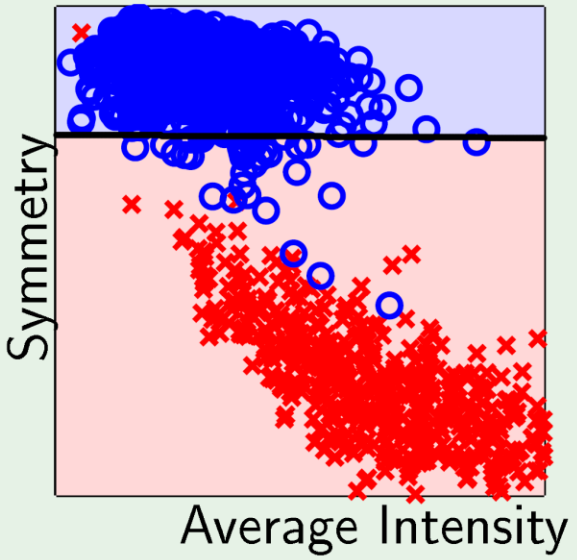


How PLA work

Evolution of E_{in} and E_{out}



Final perceptron boundary



Linear Classification Model

- Let now assume $\mathcal{X} = \{1\} \times \mathbb{R}^d,\ \mathcal{Y} = \{-1, +1\}\ and\ f: \mathcal{X} \to \mathcal{Y}$

- The linear model for binary classification uses the hypothesis class $\mathcal{H}$,

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

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

- TWO MAIN 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 (ERM)

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

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 scenaries:
  1. Linearly separable Data : which means **there exist** $\mathbf{w}^*$ with $\boldsymbol{E_{in}}(\mathbf{w}^*)$**=0**
  2. Non-Separable data

- Scenary-1:
  - We already saw the Perceptron Learning Algorithm (PLA) (remember ?)

- PLA-Result: Let $(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_m, y_m)$ be a separable sample of examples. Let $B = \min\{\|\boldsymbol{w}\|: \forall i \in [m], y_i \boldsymbol{w}^T \boldsymbol{x}_i \geq 1, w \in R^d\}$ and let $R = \max_i \|\boldsymbol{x}_i\|$. Then the PLA stops after at most $(RB)^2$ iterations, and when is stops it holds that $\forall i \in [m], y_i \boldsymbol{w}^T(t) \boldsymbol{x}_i \geq 0$.

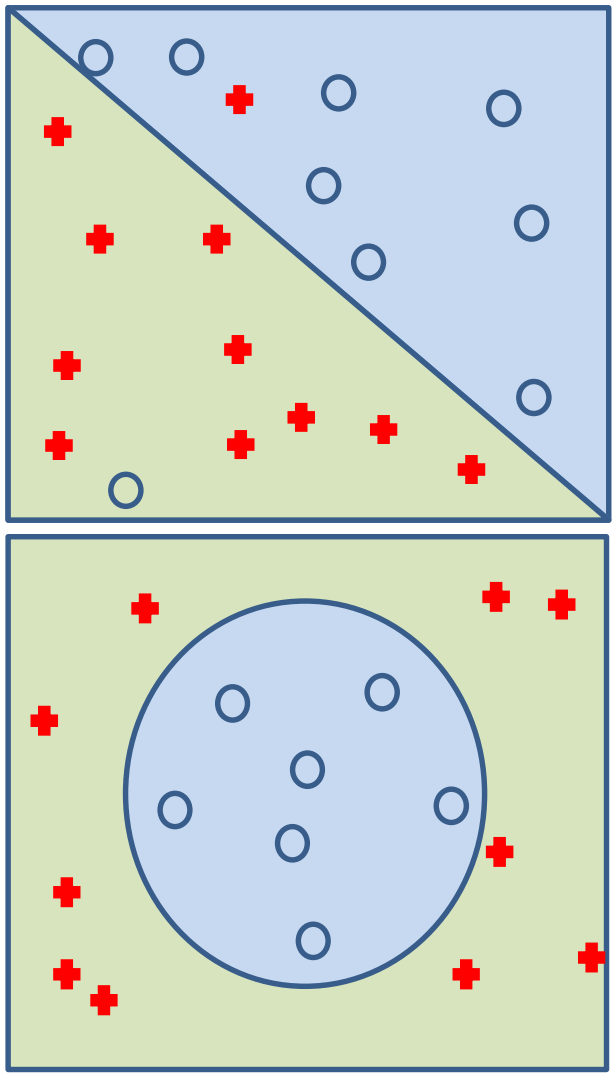
- The convergence rate depend on the parameter B. But this value can be exponentially large in d.

- As far as PLA is concerned linear separability is a property of the data $\mathcal{D}$ not the target f.
  - A linear separable $\mathcal{D}$ could have been generated either from a linear separable target, or (by chance) from a non separable target

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

Non-Separable data: Two different scenaries:

- 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

$$\underbrace{\min_{w \in R^{d+1}} \frac{1}{N} \sum_{n=1}^{N} [\![\text{sign}(\mathbf{w}^T \mathbf{x}_n) \neq y_n]\!]}_{E_{in}}$$

But this problem is known to be a NP-hard combinatorial problem in the general case.

We will approximate the problem with the Pocket Algorithm

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

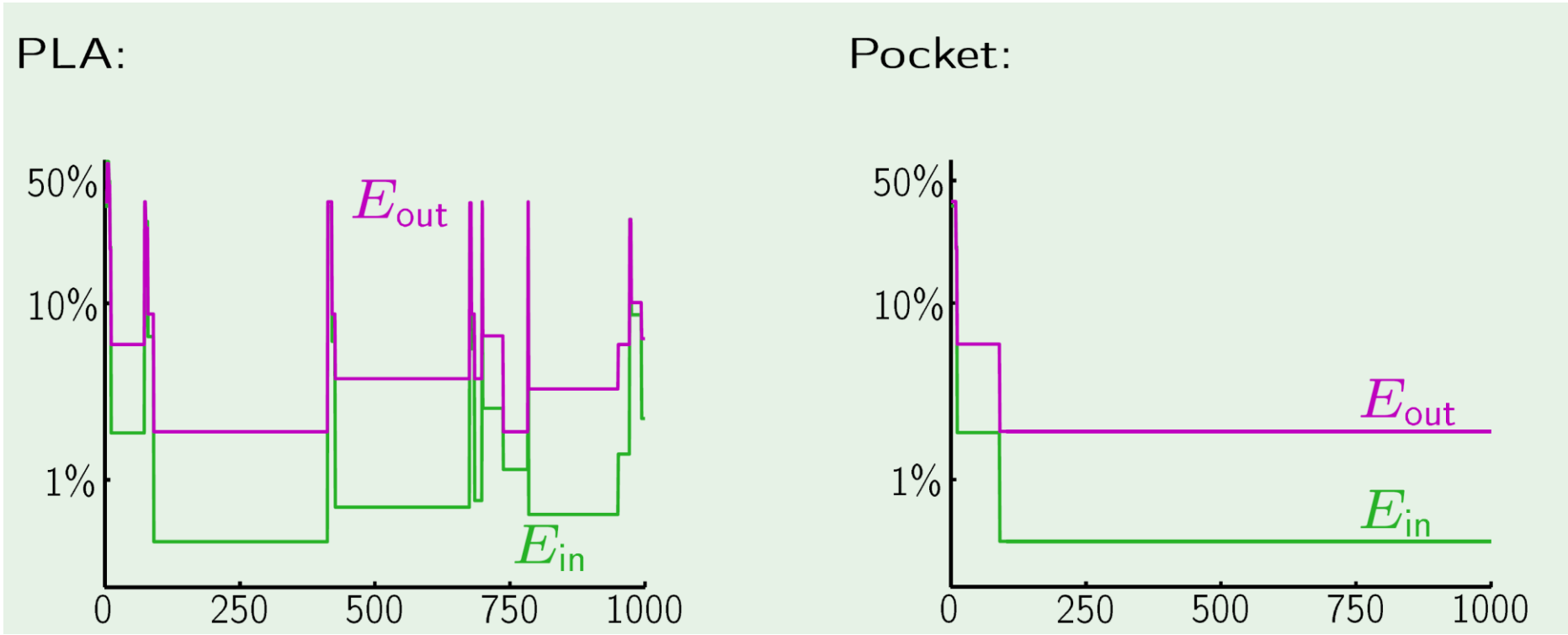
Non-Separable Case: noisy labels

- 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,…,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 $\hat{\mathbf{w}}$ in terms of E_{in}(**w**(t+1)) , set $\hat{\mathbf{w}}$= **w**(t+1)
  6. **Return** $\hat{\mathbf{w}}$

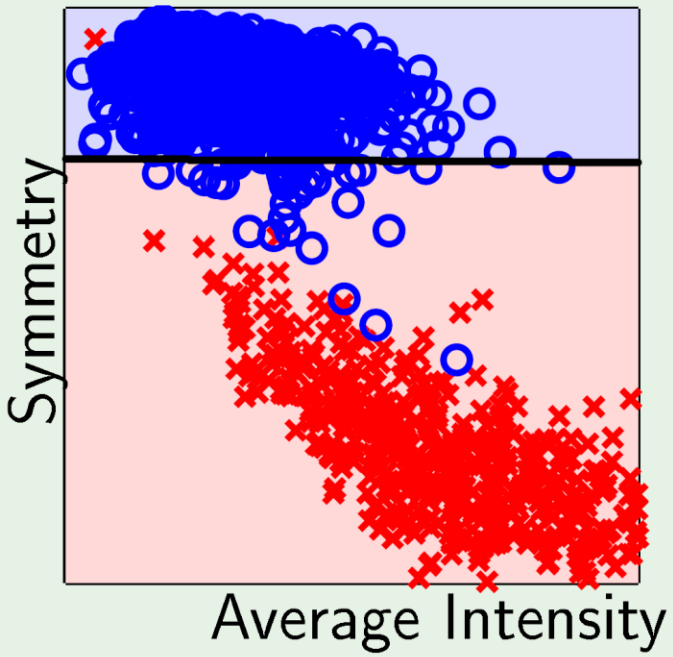
- The pocket algorithm has a clear efficiency penalty in point 4.
- But it is guaranteed to get a good solution after a fixed large number of updatings.

The ‘Pocket’ Algorithm

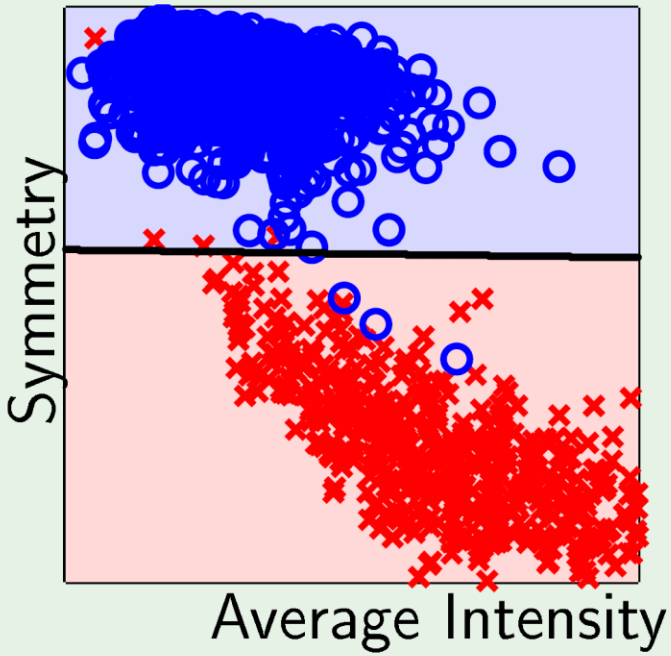


Classification boundary – The PLA vs Pocket

PLA:



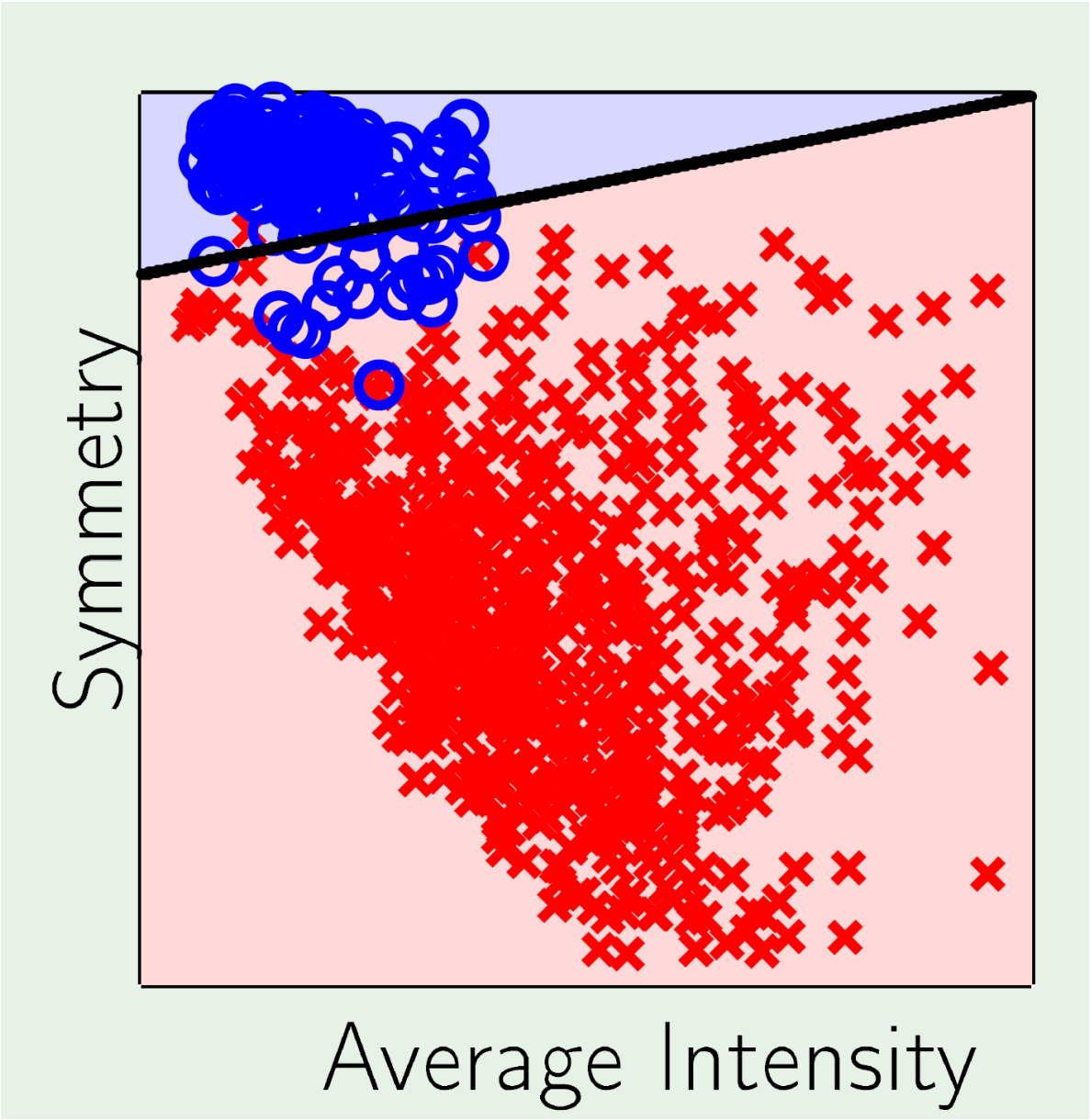
Pocket:



Linear Regression for Classification

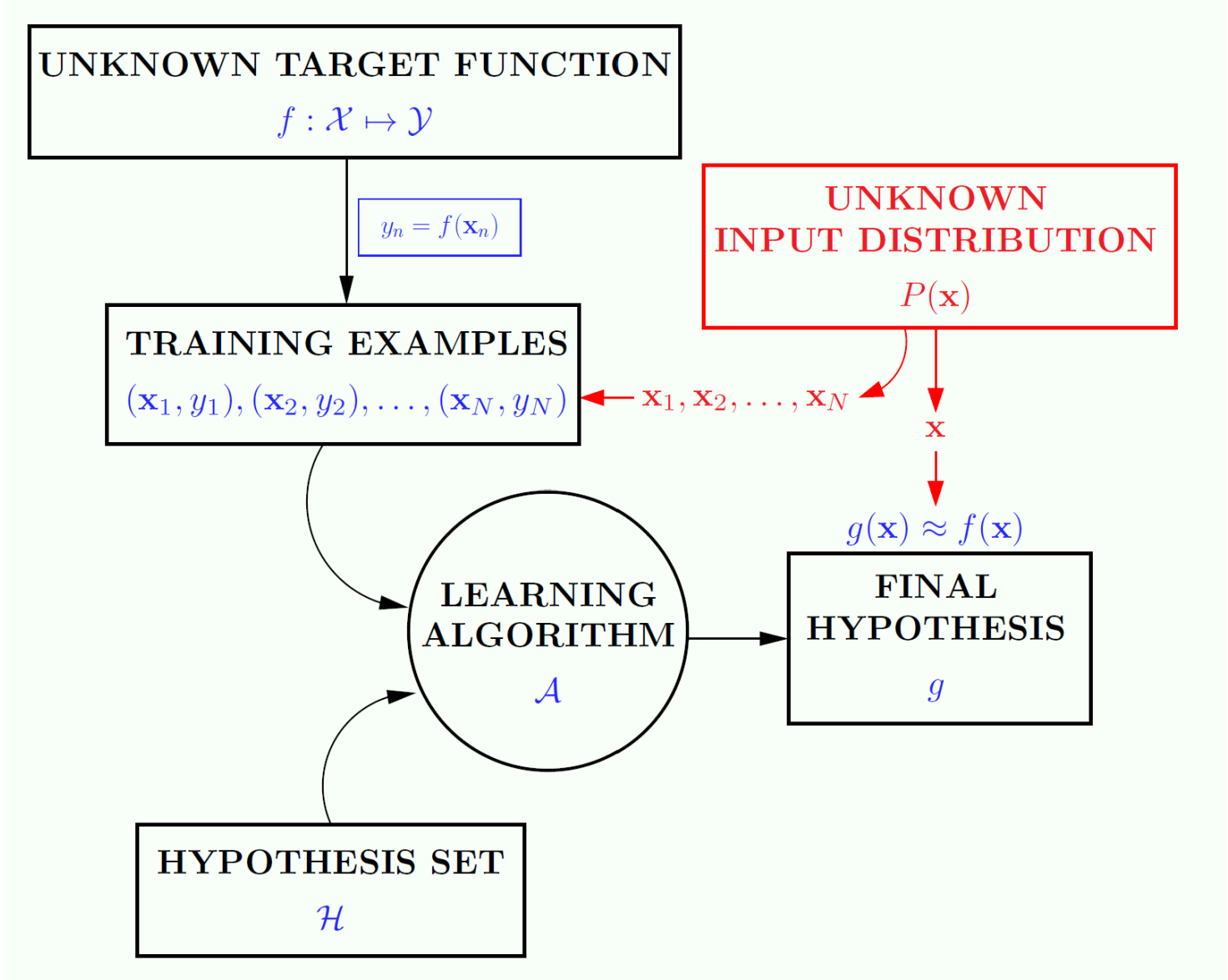
- Linear regression learns a real function $y = f(\mathbf{x}) \in \mathbb{R}$
- Binary-valued functions are also real valued! $\pm 1 \in \mathbb{R}$
- Use linear regression to get $\mathbf{w}$ where $\mathbf{w}^{\mathrm{T}}\mathbf{x}_n \approx y_n = \pm 1$
- In this case the $\mathrm{sign}\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}_{\mathrm{n}}\right)$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

Linear Regression Boundary



Noisy labels:
A general setup

Current Learning Setup



Noisy Samples

- A realistic scenary 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 is 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 $y = f(x)$ (deterministic) we now use

$$\mathrm{P}(\mathrm{y}|\mathbf{x})$$

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

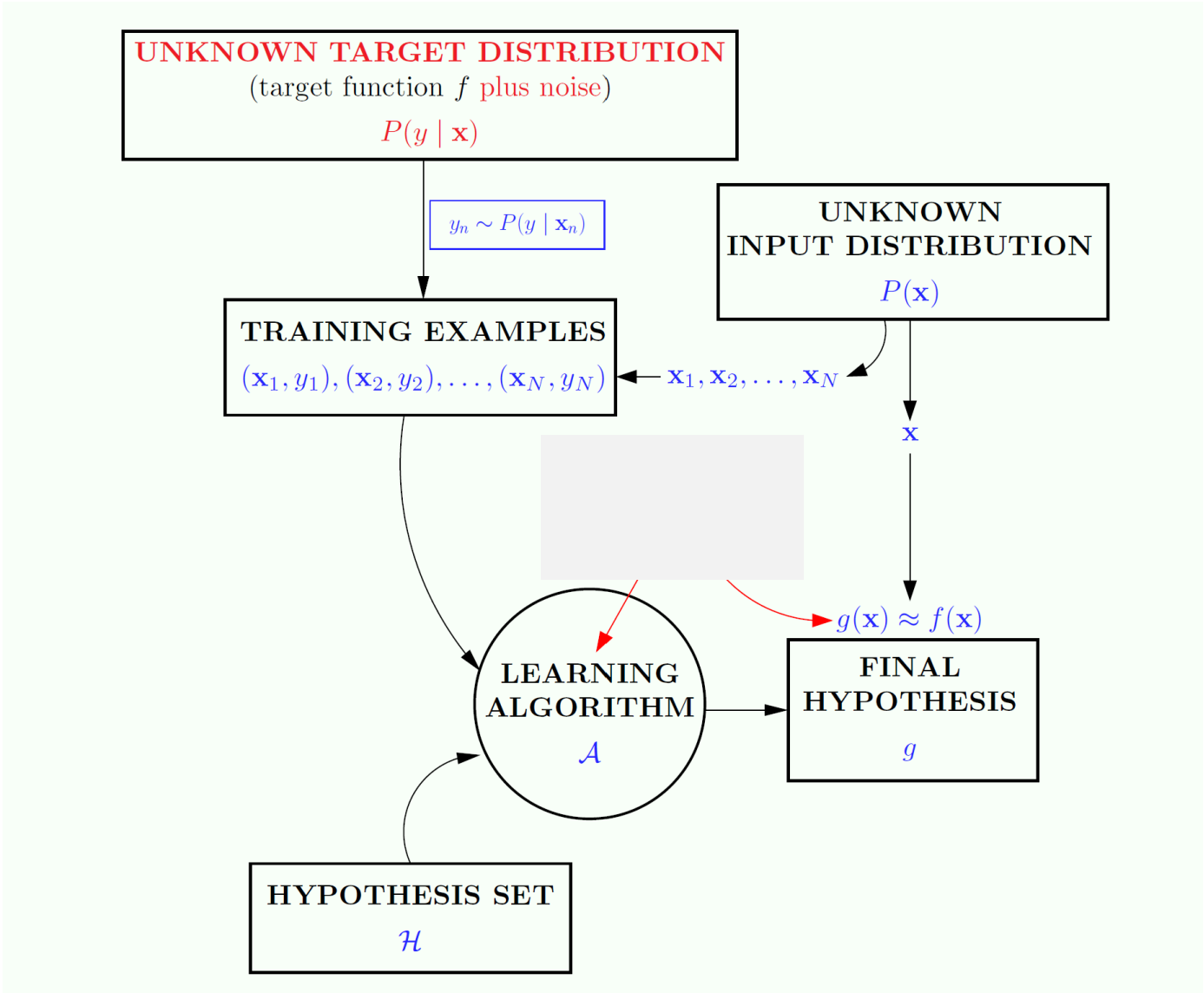
$$\mathrm{P}(\mathbf{x})\mathrm{P}(\mathrm{y}|\mathbf{x})$$

- **Noisy target** is a deterministic target $f(\mathrm{x}) = \mathrm{E}(y|\mathbf{x})$ plus noise $y - f(x)$

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

$\mathrm{P}(\mathrm{y}|\mathbf{x})$ is zero except in $y = f(x)$

Updated Learning Setup



PROBABILITY ESTIMATION

Logistic Regression (LGR)

- Now the unknown labeling function is

$$f: \mathcal{X} \to [0,1] \text{ such that } f(\boldsymbol{x}) = \mathbb{P}(y = 1|\boldsymbol{x}),$$

- But the values, we are given are deterministic (± 1) – labels of the classes

- As the (± 1) – labels are highly correlated with the function *f,* we choose as $h(\boldsymbol{x}) = \boldsymbol{\sigma}(\boldsymbol{w}^T \boldsymbol{x})$ the $\boldsymbol{w}$-vector that maximizes the probability of the simple $\boldsymbol{s} = \{(\boldsymbol{x_1}, y_1), \cdots, (\boldsymbol{x_N}, y_N),\}$ given by

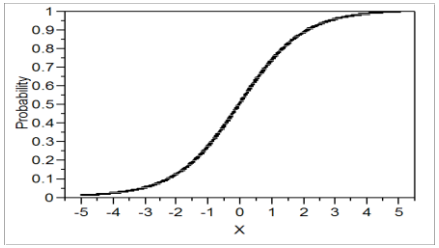
$$L(\boldsymbol{s}) = \prod_{i=1}^{N} P(y_i|\mathbf{x}_i)$$

i.e., we try to find a function h given the same (± 1) – labels on the sample points $\mathbf{x}_i$

Logistic Regression (LGR)

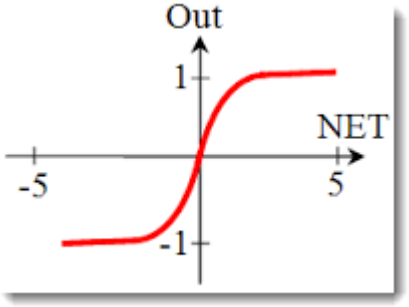
- Linear Regression: $h(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$,
- **Logistic** Regression: $h(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T \boldsymbol{x}) \in [0,1]$ or also [-1,1]
- The σ-function is called *logistic function (sigmoide):*

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



- Other function:

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



- Some advantages appear:
  - The LGR output can be considered as a classification probability.
  - It allows us some flexibility in order to assign samples to the labels

Logistic Regression notation

BINARY CASE:

- $h(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{x}) = \frac{e^{\boldsymbol{w}^T\boldsymbol{x}}}{1+e^{\boldsymbol{w}^T\boldsymbol{x}}}$ C1 probability

  $1 - h(\boldsymbol{x}) = \sigma(-\boldsymbol{w}^T\boldsymbol{x}) = \frac{1}{1+e^{\boldsymbol{w}^T\boldsymbol{x}}}$ C2 probability

The ratio $\frac{h(\boldsymbol{x})}{1-h(\boldsymbol{x})}$ is called "odds"

- $\ln(\text{odds}) = \ln\left(\frac{h(\boldsymbol{x})}{1-h(\boldsymbol{x})}\right) = \ln\left(e^{\boldsymbol{w}^T\boldsymbol{x}}\right) = \mathbf{w}^{\mathrm{T}}\mathbf{x}$ (función logit)

- **The logit function can be any regression model**

Error Measure (Loss)

- Formally, let's try to learn the target function $f(\mathbf{x}) = \mathbb{P}[y = +1|x]$

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

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

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

$$L(\mathbf{w}) = \prod_{i=1}^{N} P(y_i|\mathbf{x}_i) = \prod_{i=1}^{N} \sigma(y_i\,\mathbf{w}^{\mathrm{T}}\mathbf{x}_i) = \prod_{j=1}^{N_1} \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_j) \prod_{k=1}^{N_{-1}} \sigma(-\mathbf{w}^{\mathrm{T}}\mathbf{x}_k)$$

Learning Criteria: ML

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

Equivalently Minimize ERM:

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

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

$$\nabla_{\boldsymbol{w}} E_{\text{in}}(\mathbf{w}) = \frac{\partial}{\partial \mathbf{w}}\left(\frac{1}{N}\sum_{i=0}^{N}\ln\big(1+e^{-y_i\mathbf{w}^{\mathrm{T}}\mathbf{x}_i}\big)\right) = \frac{1}{N}\sum_{i=0}^{N} -y_i\mathbf{x}_i\frac{e^{-y_i\mathbf{w}^{\mathrm{T}}\mathbf{x}_i}}{1+e^{-y_i\mathbf{w}^{\mathrm{T}}\mathbf{x}_i}}$$

$$= \frac{1}{N}\sum_{i=0}^{N} -y_i\mathbf{x}_i\sigma(-y_i\mathbf{w}^{\mathrm{T}}\mathbf{x}_i)$$

Multilabel classification

- Binary logistic regression can be generalized to cope with k labels
  - In the binary approach we maximize the probability of the binary labelling given by the likelihood of N Bernoulli variables {0,1}

$$\mathrm{L}(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N) = \prod_{i=1}^{N} \sigma(\boldsymbol{w}^T \boldsymbol{x}_i)^{[\![y_i==1]\!]} (1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}_i))^{[\![y_i==0]\!]} = \prod_{i=1}^{N} \prod_{k=0}^{1} \sigma\,(\boldsymbol{w}^T \boldsymbol{x}_i)^{y_{ik}}$$

- 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,..,1,..,0] or [-1,-1,…,1,..,-1]

The sample likelihood (K clases)

Let $(\mathbf{x}, \boldsymbol{y})$ denote a single sample, then **assuming conditional independence**

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

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

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

- The likelihood is defined by

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

Multilabel classification: SoftMax

- $t_{nk} = \sigma(\mathbf{w}_k^T \mathbf{x}_n),\ \sum_{k=1}^{K} t_{nk} = 1,$
- $E(\mathbf{w}_1, \mathbf{w}_{2,} \cdots, \mathbf{w}_K) = -\ln L(Y|\, \mathbf{w}_1, \mathbf{w}_{2,} \cdots, \mathbf{w}_K) = -\sum_{n=1}^{N} \sum_{k=1}^{K} y_{nk} \ln t_{nk}$
- $\nabla_{\mathbf{w}_j} E(\mathbf{w}_1, \mathbf{w}_{2,} \cdots, \mathbf{w}_K) = \sum_{n=1}^{N} (t_{nj} - y_{nj}) \mathbf{x}_n$
- **SOFTMAX**: from each sample $\mathbf{x}$ a vector of probabilities is computed using the estimated vector $\mathbf{w}_1, \mathbf{w}_{2,} \cdots, \mathbf{w}_K$

$$P(\mathrm{C}_j|\mathbf{x}) = \frac{\exp(\mathbf{w}_j \mathbf{x})}{\sum_k \exp(\mathbf{w}_k \mathbf{x})}, \qquad \mathrm{j} = 1, \ldots, \mathrm{K}$$

Decision Rule: To assign x to $\mathrm{C_j}$ if $P(\mathrm{C}_j|\mathbf{x}) = \max_k P(\mathrm{C}_k|\mathbf{x})\,,\ k = 1, .., K$

ERM: 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: $ERM_{\mathcal{H}}(\mathcal{D}) = \text{argmin}_h \left\{ \frac{1}{N} \sum_{i=1}^{N} \text{I}[y_i \neq h(\mathbf{x}_i)] \right\}$
  - Regression problems: $ERM_{\mathcal{H}}(\mathcal{D}) = \text{argmin}_h \left\{ \frac{1}{N} \sum_{i=1}^{N} \left(y_i - h(\boldsymbol{x}_i)\right)^2 \right\}$
  - Logistic regression: $ERM_{\mathcal{H}}(\mathcal{D}) = \text{argmin}_h \left\{ \frac{1}{N} \sum_{i=1}^{N} \ln(1 + e^{-y_i h(\boldsymbol{x}_i)}) \right\}$

- 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($\mathcal{Y} \mid \mathcal{X}$)

$$ERM_{\mathcal{H}}(\mathcal{D}) \in \underset{h \in \mathcal{H}}{\text{argmin}}\, E_{in}(h)$$

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

SGD+ERM: A new induction rule

- We can apply SGD to solve any optimization problem where the gradient can be computed.

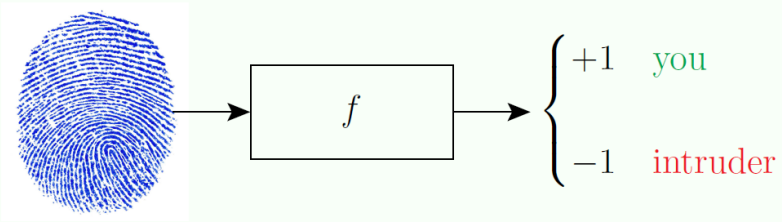
- Example.1 : <u>Regression</u>
  - The error function is $\mathrm{e_n}(\mathbf{w}) = (y_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{x}_n)^2$
  - SGD can solve the problem iterating on bunch of examples.
  - Furthermore, linear regression convergence can be speeded using the second derivative and the **Newton method**.

- Example.3 : <u>Perceptron (PLA)</u>
  - In PLA we minimize the error according to the updating rule: $\mathbf{w}(\mathrm{t}+1) = \mathbf{w}(\mathrm{t}) + \mathrm{y_i}\mathbf{x_i}$
  - This rule can be seen as a SGD rule with η=1 and gradient given by $\mathrm{y_i}\mathbf{x_i}$
  - Can be verified that the function $\mathrm{e_n}(\mathbf{w}) = \max(0, -y_n\mathbf{w}^{\mathrm{T}}\boldsymbol{x}_n)$ provides the gradient

- Example.2 : <u>Logistic Regression (LGR)</u>
  - Now the error function is $\mathrm{e_n}(\mathbf{w}) = \ln(1 + e^{-y_n\mathbf{w}^{\mathrm{T}}\boldsymbol{x}_n})$
  - For $\mathbf{w}$ large $\mathrm{e_n}(\mathbf{w}) \approx \max(0, -y_n\mathbf{w}^{\mathrm{T}}\boldsymbol{x}_n)$ making LGR+SGD equivalent to PLA

Error Costs

Error measure user-specified



Two types of error.

		f: $+1$	f: -1
h	$+1$	no error	**false accept**
	-1	**false reject**	no error

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

		f: $+1$	f: -1
h	$+1$	0	1
	-1	**10**	0

Supermarket

		f: $+1$	f: -1
h	$+1$	0	**1000**
	-1	1	0

CIA

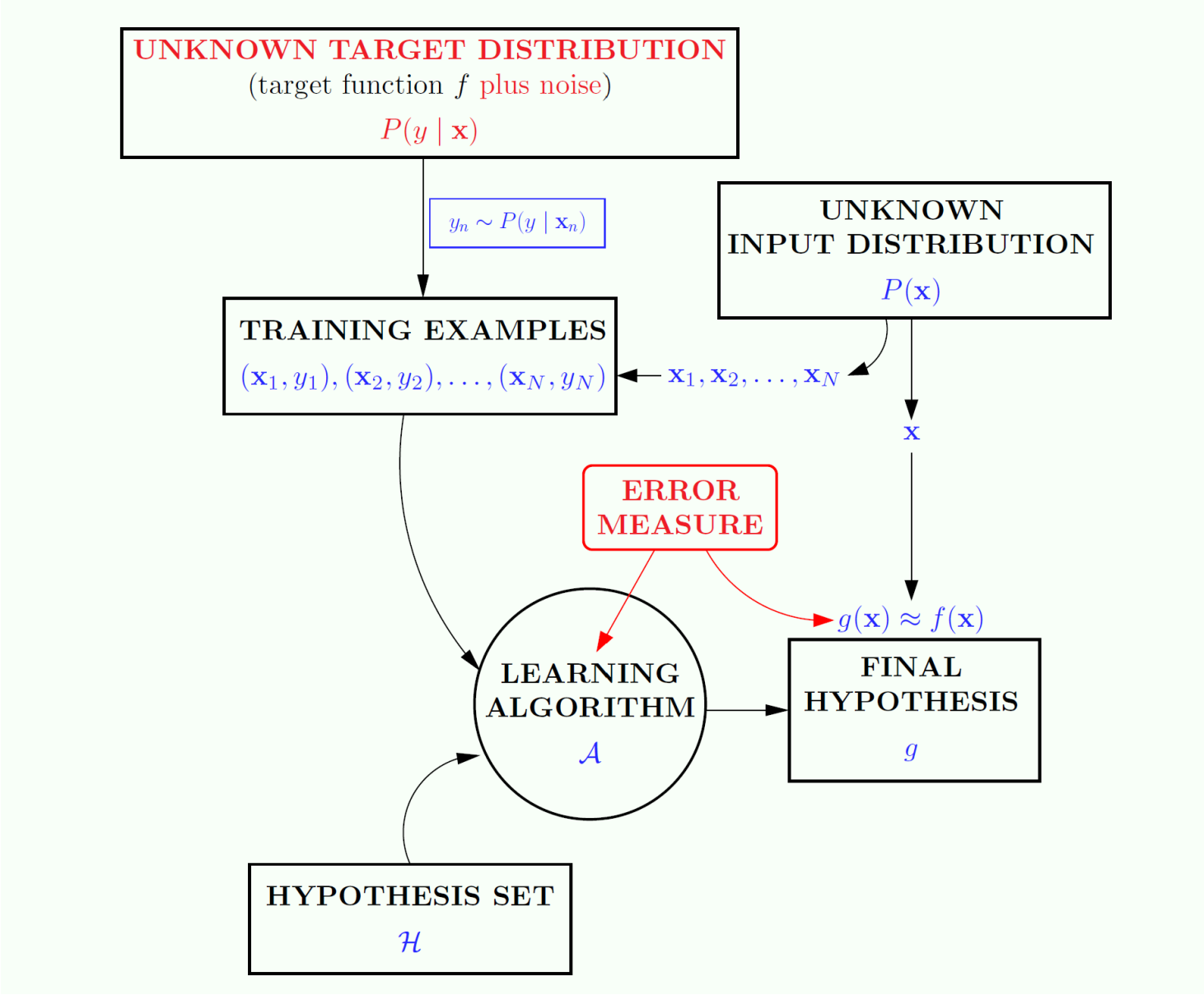
Take Away

Error measure is specified by the user.

If not, choose one that is

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

Learning Setup with Error Measure and Noisy Target



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

- Both distribution are relevant in the learning process

- The target distribution $P(y|\mathbf{x})$ is what we want to learn
- The input distribution $P(\mathbf{x})$ quantify the relative importance of **x**

- Let´s consider the expression $P(\mathbf{x}, y) = P(\mathbf{x})P(y \mid \mathbf{x}) = P(\mathbf{x} \mid y)P(y)$
  - Clearly, learning is over once we know $P(\mathbf{x}, y)$

- There are two ways of computing $P(y|\mathbf{x})$

  - To learn $P(y|\mathbf{x})$ directly (Discriminative approaches)
  - To learn $P(y|\mathbf{x})$ through $P(\mathbf{x} \mid y)$ 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 $\mathcal{X}\times\{0,1\}$, the best label predicting function from $\mathcal{X}$ to $\{0,1\}$ will be

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

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 this 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)

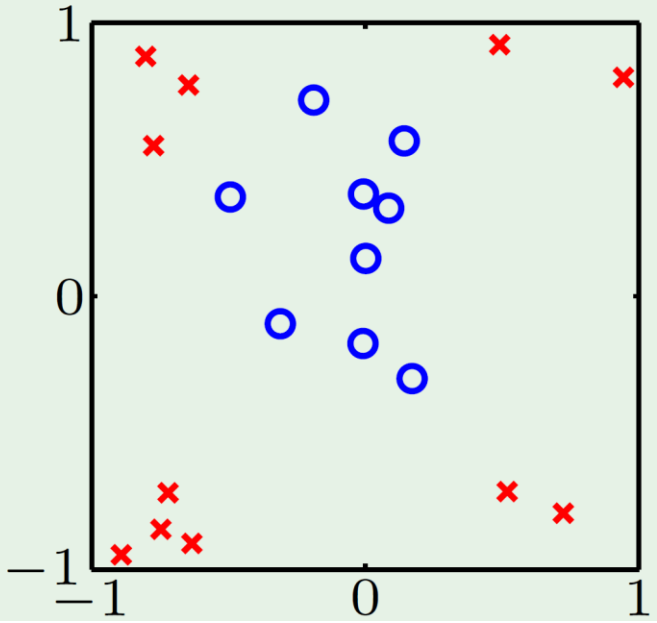
"To solve a problem never solve a more complex one as intermediate step" (Vapnik 1995).

Adding functions to $\mathcal{H}$:

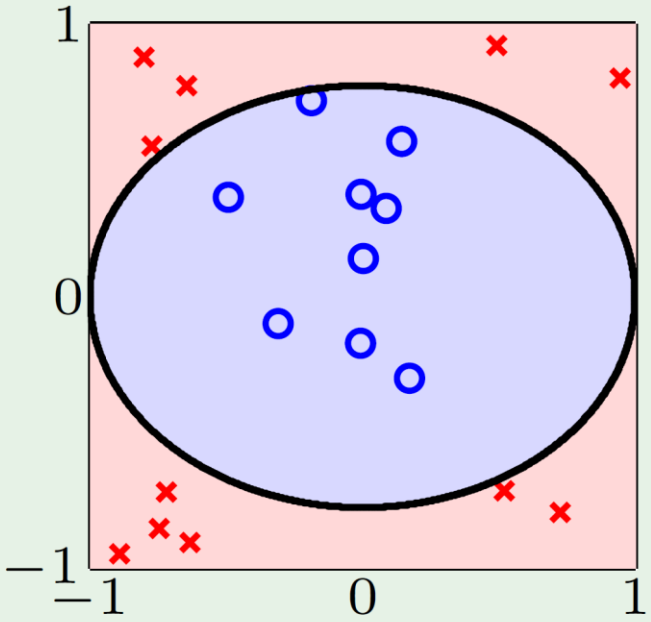
Non linear Transformations

Linear predictors are limited

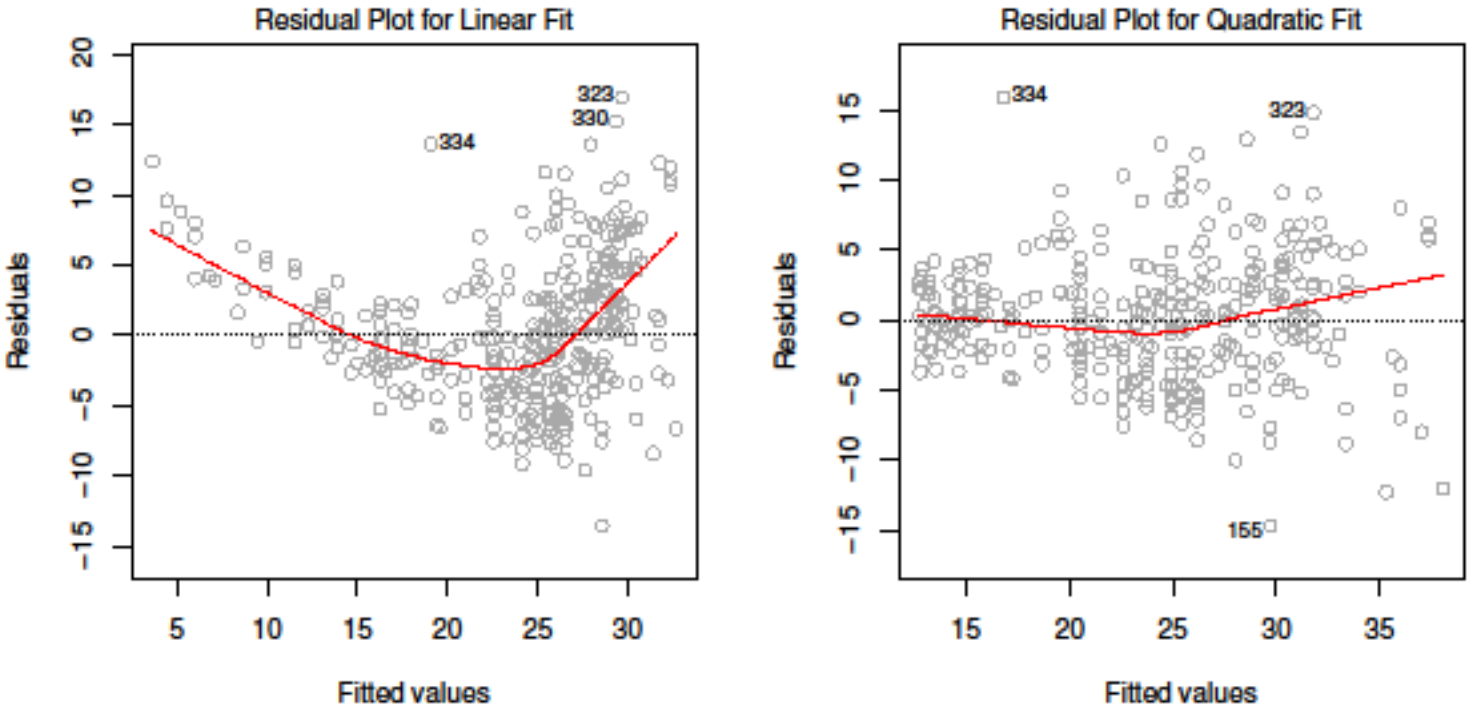
Data:



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 $[\![x_i > 5]\!]$ and $[\![x_i < 1]\!]$ are better features

How can we do this with linear models ?

- Linear regression implement:

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

- Linear classification implement:

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

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, .., x^n,
  - Transcendent functions: log(x), sin(x), $x^{1/2}$,…,
  - 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 !!**

Polynomial Regression

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

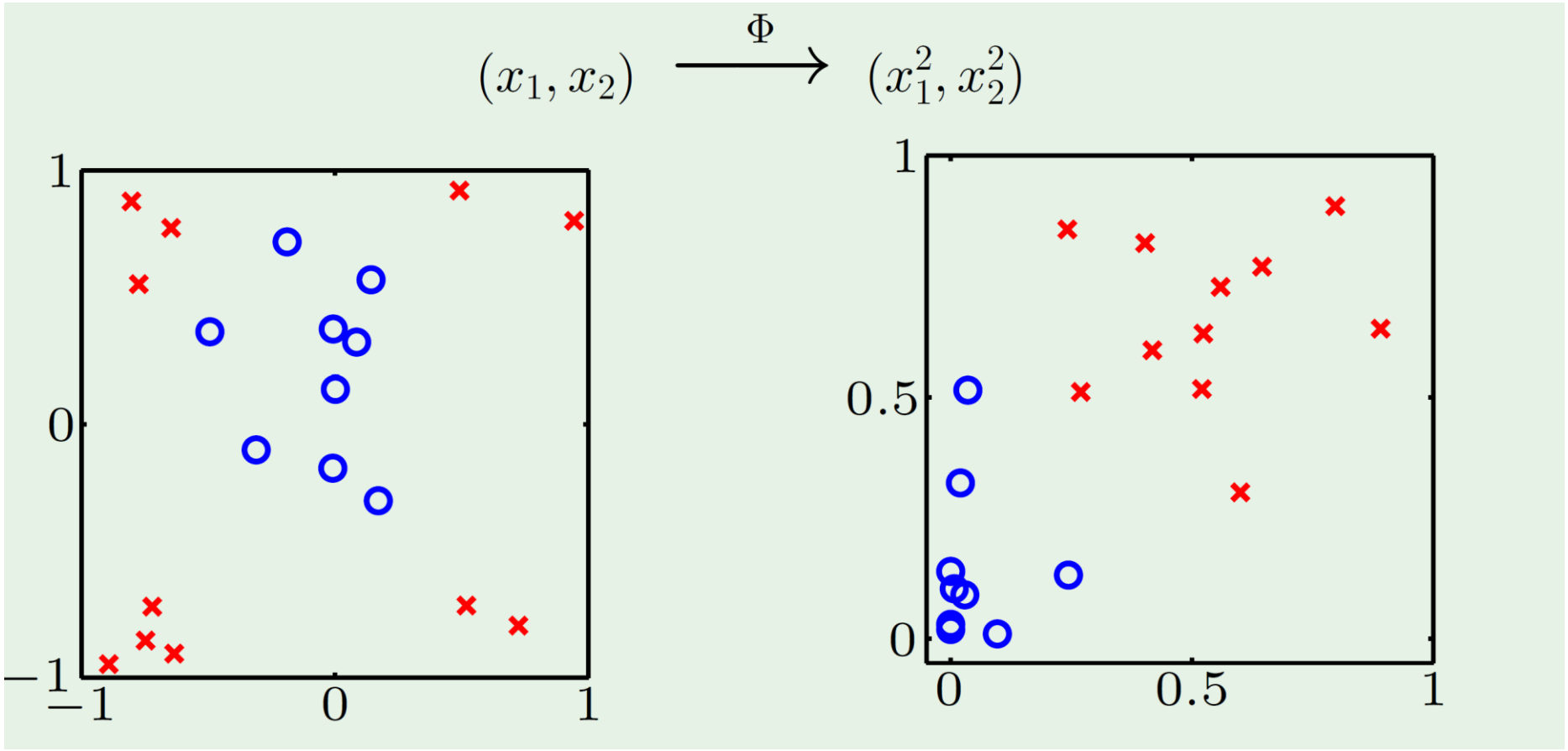
$$y = \beta_0 + \beta_1 X + \beta_2 X^2 \qquad \text{(quadratic model)}$$
$$y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 \text{ (cubic model)}$$

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

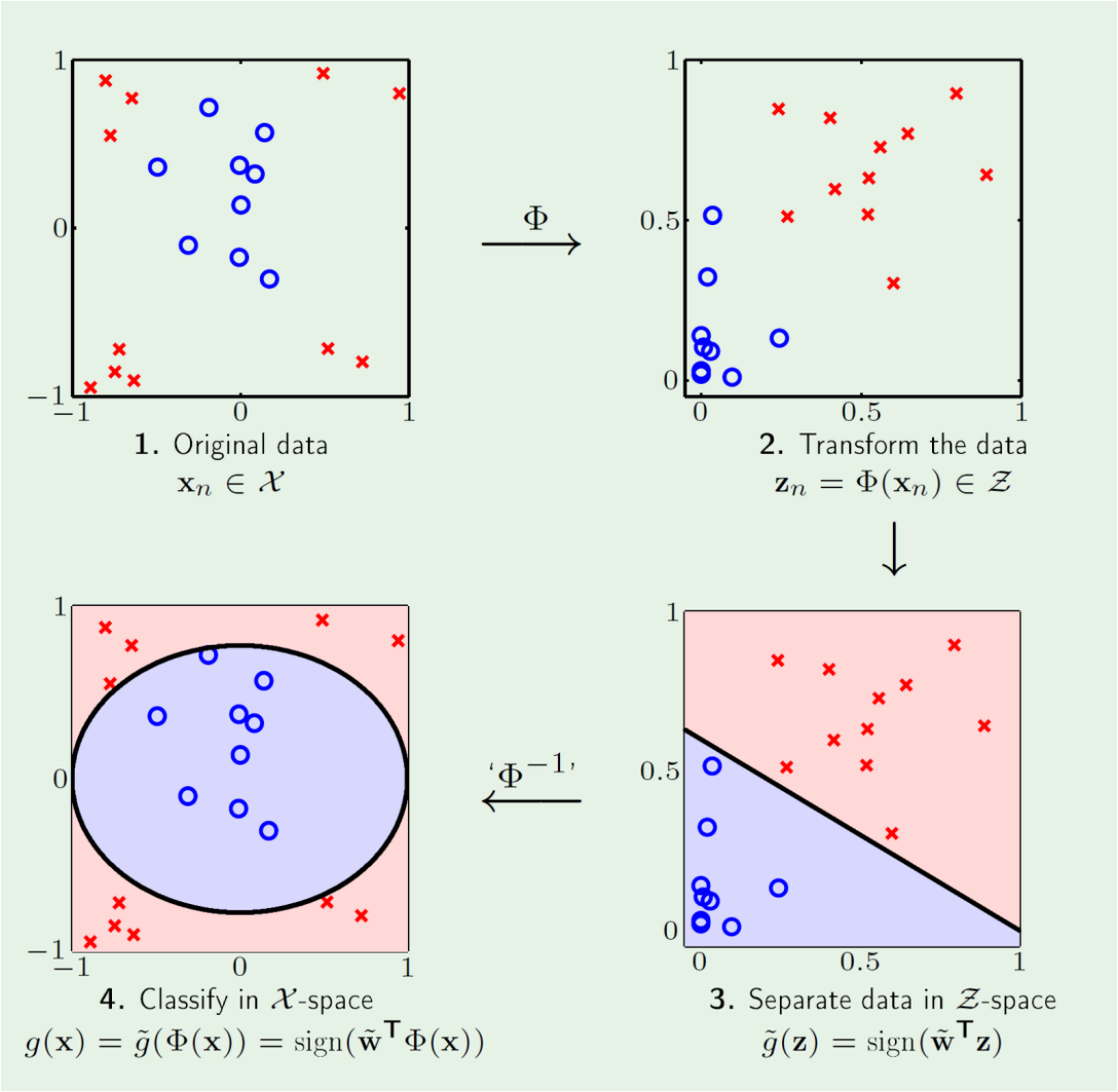
$$\mathrm{X} = \begin{bmatrix} 1 & x_1 & \cdots & x_1^p \\ 1 & x_2 & \cdots & x_2^p \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & \cdots & x_N^p \end{bmatrix}$$

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

Transforn the data nonlinearly



Nonlinear Transformations



What transforms to what

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

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

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

No weights in $\mathcal{X}$ $\qquad$ $\tilde{\mathbf{w}} = (w_0, w_1, \cdots\cdots\cdots, w_{\tilde{d}})$

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

Generalized Error/Loss Functions

- Let denote $l(h, z)$ a general error/loss function of a hypothesis h on an example $\boldsymbol{z} = (\phi(\boldsymbol{x}), y)$

$$l(h, \boldsymbol{z}) \text{ represent one of theses functions } \{(h(\boldsymbol{z}) - y)^2, \mathbb{1}(h(\boldsymbol{z}) \neq y), etc\}$$

$$\ell_{0-1}(h, \boldsymbol{z}) \stackrel{\text{def}}{=} \begin{cases} 0 & if \quad h(\phi(\boldsymbol{x})) = y \\ 1 & if \quad h(\phi(\boldsymbol{x})) \neq y \end{cases}$$

0-1 loss

$$\ell_{sq}(h, \boldsymbol{z}) \stackrel{\text{def}}{=} (h(\phi(\boldsymbol{x})) - y)^2$$

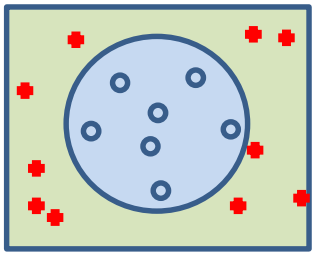
Square loss

- The Risk Function is used to measure the expected error/loss of a hypothesis with respect to a distribution probability $\mathcal{P}$

$$L_{\mathcal{P}}(h) \stackrel{\text{def}}{=} \mathbb{E}_{z \sim \mathcal{P}}[l(h, \boldsymbol{z})] \stackrel{\text{def}}{=} E_{out}(h)$$

  - That definition is an average on all posssible examples (risk(h) = average loss)
  - This expresion is applicable in any learning setting

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 $h(\boldsymbol{x}) = \text{sign}((1)\, x_1^2 + (1)x_2^2 - 0.6)$ 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 $\boldsymbol{z}$: $\tilde{h}(\boldsymbol{z}) = \text{sign}\big((1)\, z_1 \; + (1)z_2 \; - 0.6\big)$ Fine!!

$$\tilde{h}(\boldsymbol{z}) = \tilde{h}(\Phi(\boldsymbol{x})) = h(\boldsymbol{x})$$

- In real problems we have to **fix the transform before seeing the data**. So, we do not know the better specific transforms.

Computation and Generalization

- Let denote by Φ_Q the *Q-th order polynomial transform*
  - $\Phi_4(\boldsymbol{x}) = (1, x_1, x_2, x_1^2, x_2^2, x_1x_2, x_1^3, x_2^3, x_1x_2^2, x_2x_1^2, x_1^4, x_2^4, x_1^2x_2^2, x_2^1x_1^3, x_1^1x_2^3)$

- A larger *Q* provides a larger flexibility in terms of the shape of the decision boundary but there is a price to pay.

1. Computation is an issue because the feature transform Φ_Q maps x (the initial vector) to $\boldsymbol{d} = \frac{Q(Q+3)}{2} = \mathcal{O}(Q^2)$ dimensions, incrementing memory and computational cost.

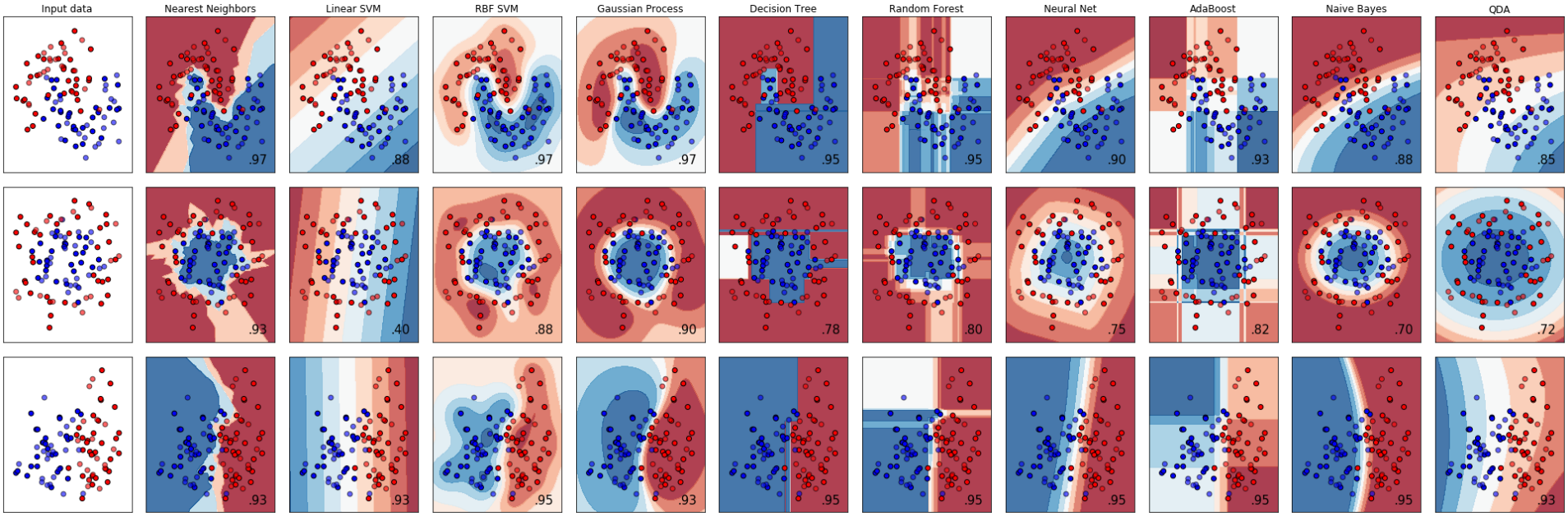
1. The higher the Q-value the higher (quadratic order) the number of samples we will need to get the same level of generalization error. (remenber $\mathcal{O}(dN^{-1}\log(N))$)

- In general when choosing the appropiate dimension for the feature transform, we must use an approximation-generalization tradeoff:

higher d better chance of being linearly separable $(E_{\text{in}} \downarrow)$ and $E_{out} \uparrow$

lower d possibly non linearly separable $(E_{\text{in}} \uparrow)$ and $\text{E}_{out} \downarrow$

Different learners ($\mathcal{A}$,$\mathcal{H}$)



2D intuition is lost in high dimensionality

Understanding E_{out}

Bias-Variance Tradeoff

- BIAS-VARIANCE decomposition

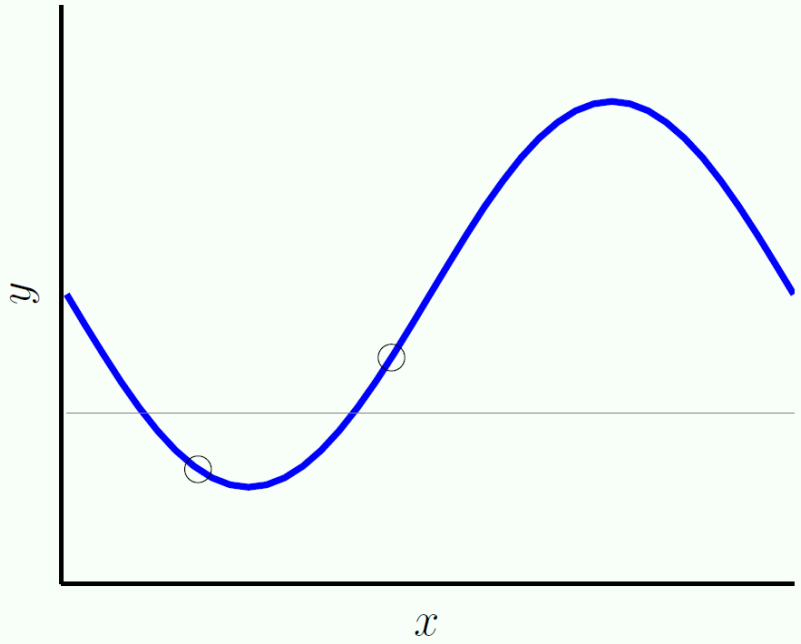
$$E_{\text{out}}\left(g^{(\mathcal{D})}\right) = \mathbb{E}_{\boldsymbol{x}}\left[\left(g^{(\mathcal{D})}(\boldsymbol{x}) - f(\boldsymbol{x})\right)^2\right]$$

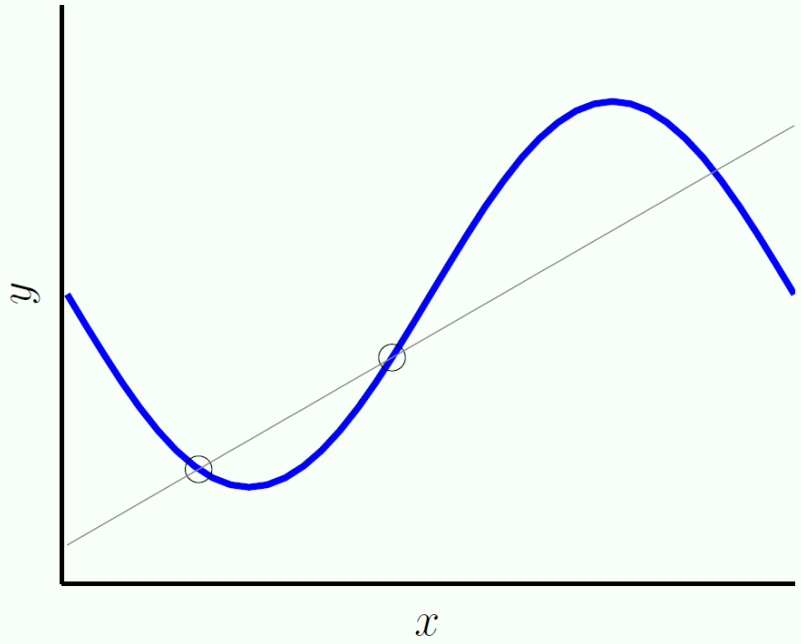
  - $\mathbb{E}_{\boldsymbol{x}}$ denotes the expected value with respect to $\boldsymbol{x}$ (based on $\mathbb{P}(\mathcal{X})$)
  - That is the Mean Squared Error (MSE) of $g^{(\mathcal{D})}$

- Bias-variance analysis is based on squared-errors measure, but applies to classification and regression.
- Bias-variance analysis takes into account $\mathcal{H}$ and $\mathcal{A}$
- Different learning algorithms $\mathcal{A}$ can have different E_{out} when applied to the same $\mathcal{H}$!!

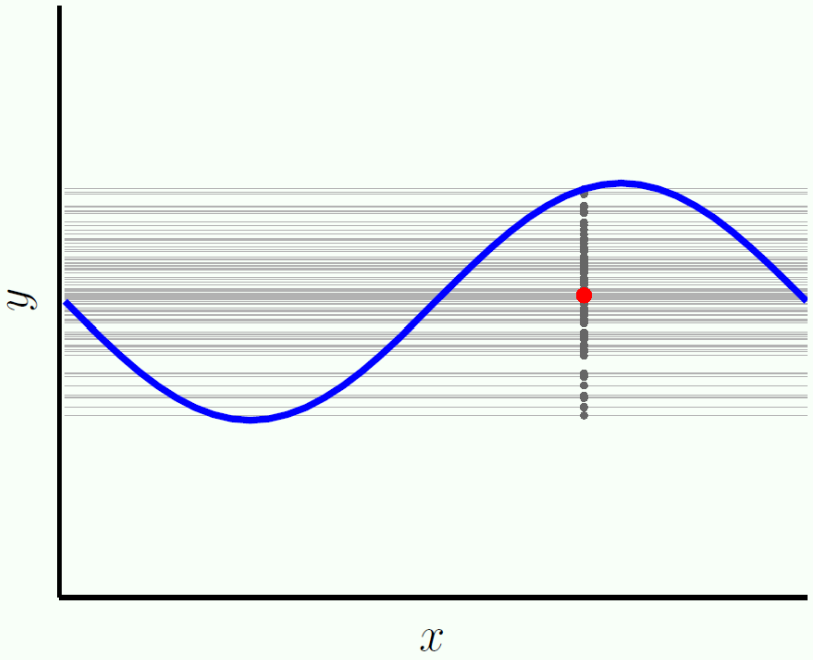
A simple learning problem

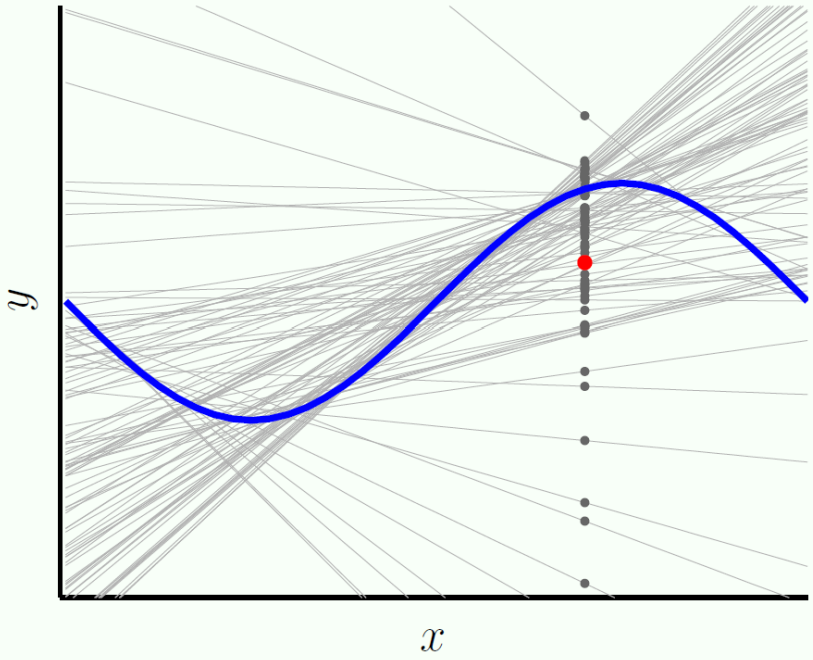
- 2 data points. 2 hypothesis sets
- $\mathcal{H}_0: h(x) = b$
- $\mathcal{H}_1: h(x) = ax + b$





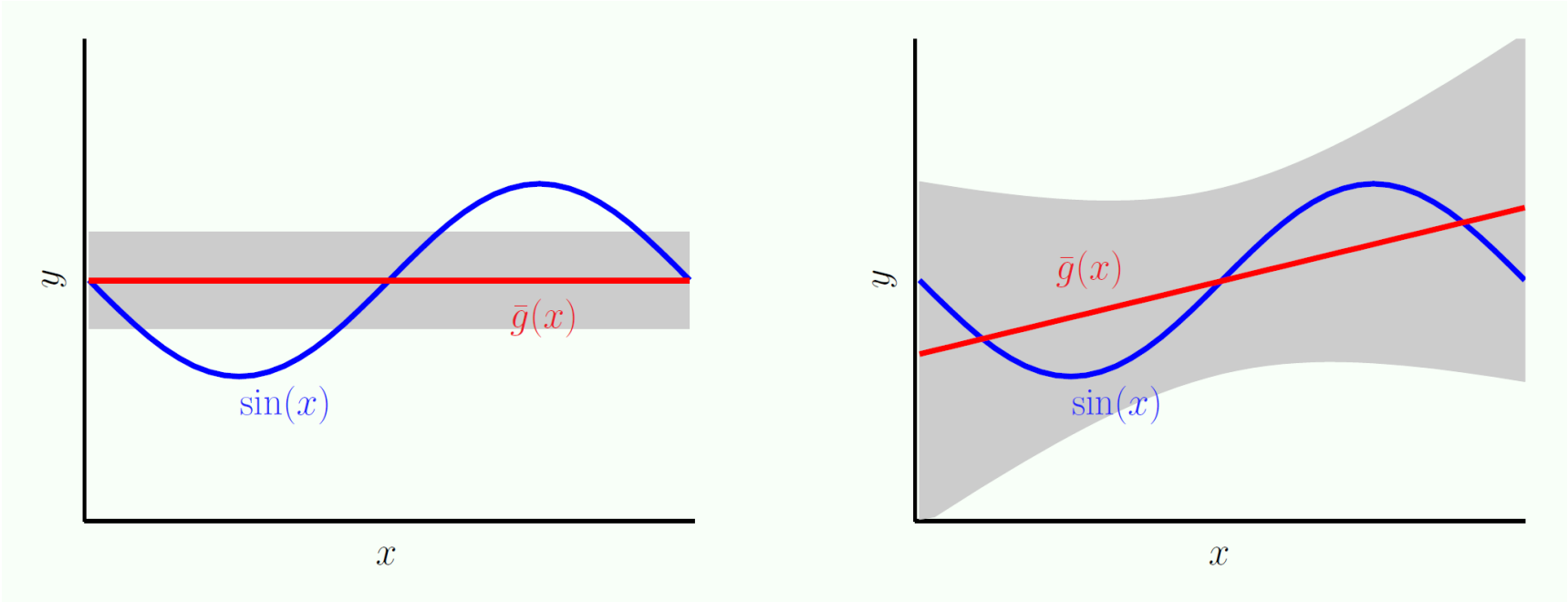
Let repeat the experiment multiples times …





- 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

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

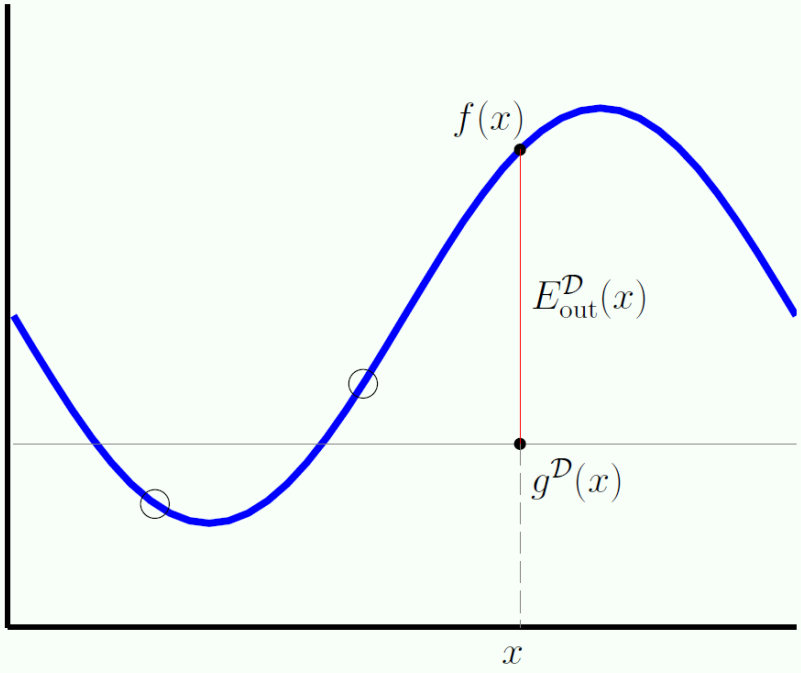
$$\bar{g}(\mathbf{x}) = \mathbb{E}_{\mathcal{D}}[g^{\mathcal{D}}(\mathbf{x})]$$

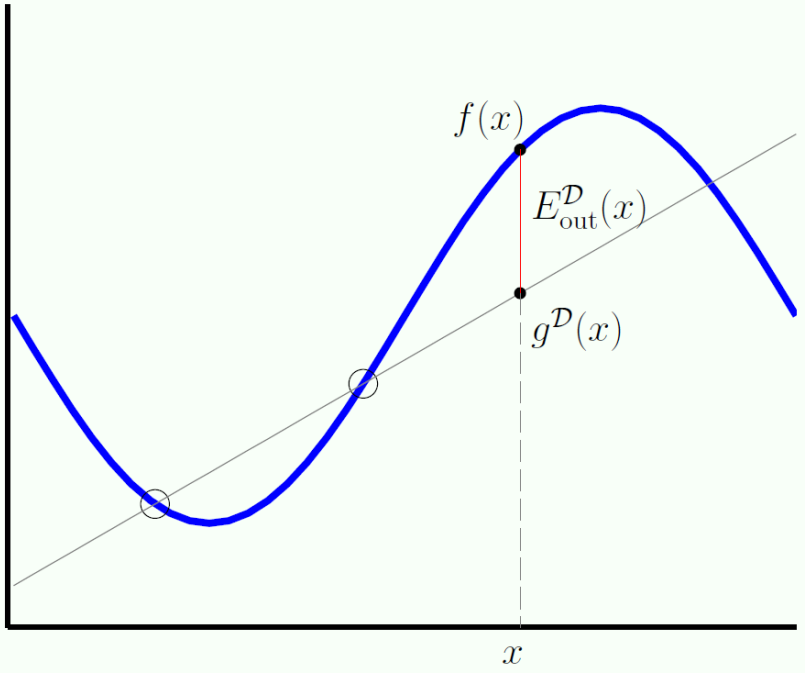
$$\approx \frac{1}{K}\left(g^1(\mathbf{x}) + g^2(\mathbf{x}) + \cdots + g^K(\mathbf{x}) \right)$$ ← the average prediction on **x**

$$\mathrm{var}(\mathbf{x}) = \mathbb{E}_{\mathcal{D}}[(g^{\mathcal{D}}(\mathbf{x}) - \bar{g}(\mathbf{x}))^2]$$

$$= \mathbb{E}_{\mathcal{D}}[g^{\mathcal{D}}(\mathbf{x})^2] - \bar{g}(\mathbf{x})^2$$ ← how variable is the prediction?

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





$E^{\mathcal{D}}_{out}(\mathbf{x}) = (g^{\mathcal{D}}(\mathbf{x}) - f(\mathbf{x}))^2$ ← squared error, a random value depending on $\mathcal{D}$

$E_{out}(\mathbf{x}) = \mathbb{E}_{\mathcal{D}}\left[E^{\mathcal{D}}_{out}(\mathbf{x})\right]$ ← expected $E_{out}(\mathbf{x})$ before seeing $\mathcal{D}$

Bias-Variance Tradeoff

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

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

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

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

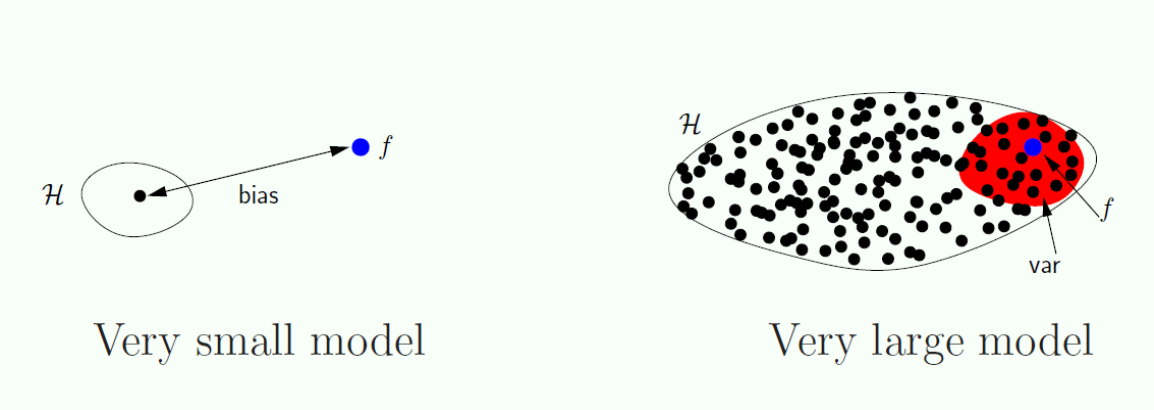
$$\mathbb{E}_{\mathcal{D}}\left[E_{out}\left(g^{(\mathcal{D})}\right)\right] = \mathbb{E}_{x}\left[\mathbb{E}_{\mathcal{D}}\left(g^{\mathcal{D}}(\boldsymbol{x})^2\right) - 2\,\tilde{g}(\boldsymbol{x})\, f(\boldsymbol{x}) + f(\boldsymbol{x})^2\right]$$

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

$$\mathbf{variance}(\boldsymbol{x}) \qquad \mathbf{bias}(\boldsymbol{x})$$

$$\mathbb{E}_{\mathcal{D}}\left[E_{out}\left(g^{(\mathcal{D})}\right)\right] = \mathbb{E}_{x}\left[\mathbf{bias}(\boldsymbol{x}) + \mathbf{variance}(\boldsymbol{x})\right] = \mathbf{bias} + \mathbf{variance}$$

Bias-Variance Tradeoff: Comments

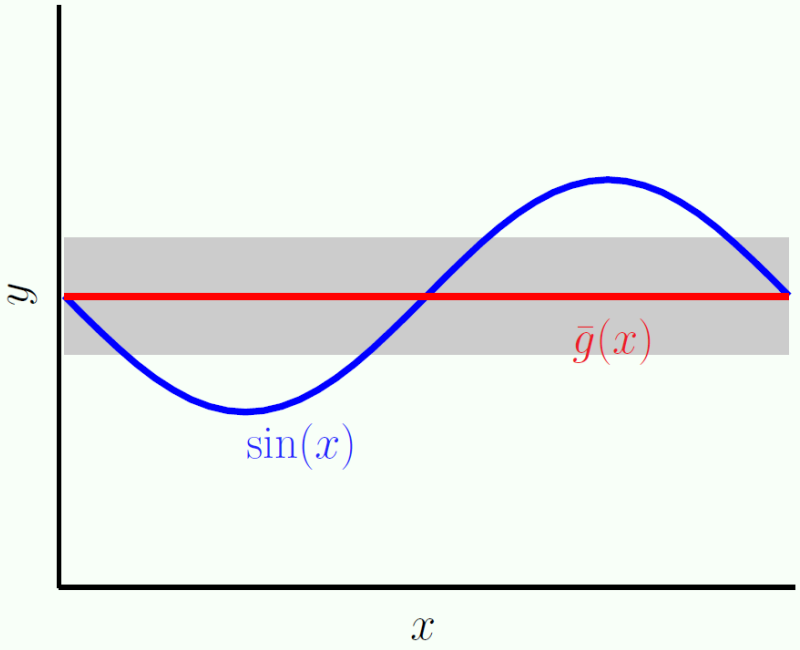


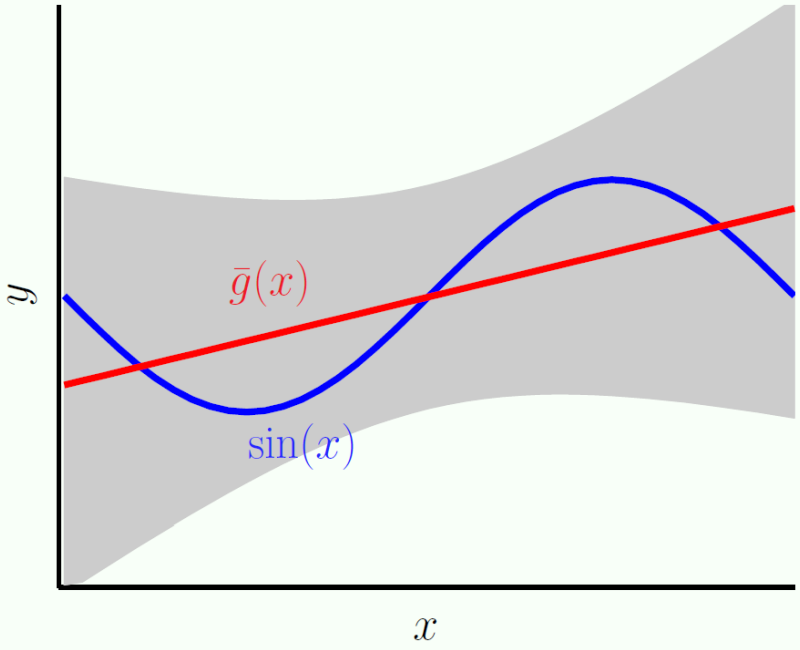
- $\mathbb{E}_{\mathcal{D}}\left[E_{out}\left(g^{(\mathcal{D})}\right)\right] = \sigma^2 + \mathbf{bias} + \mathbf{variance}$ (for noisy signals)
  - σ^2 is the variance of the noise
  - The noise is unavoidable no matter what we do, so our interest remains in bias and variance
  - Unfortunately it is impossible to compute bias and variance. Thus, the bias-variance decomposition is a conceptual tool which is heplful when it comes to developing a model.

- There are two tipical goals when we consider bias and variance:
  - To lower the variance without significatively increase the bias (1)
  - To lower the bias without significatively 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 . . .





$\mathcal{H}_0$

$\text{bias} = 0.50$

$\text{var} = 0.25$

$E_{\text{out}} = 0.75$ ✓

$\mathcal{H}_1$

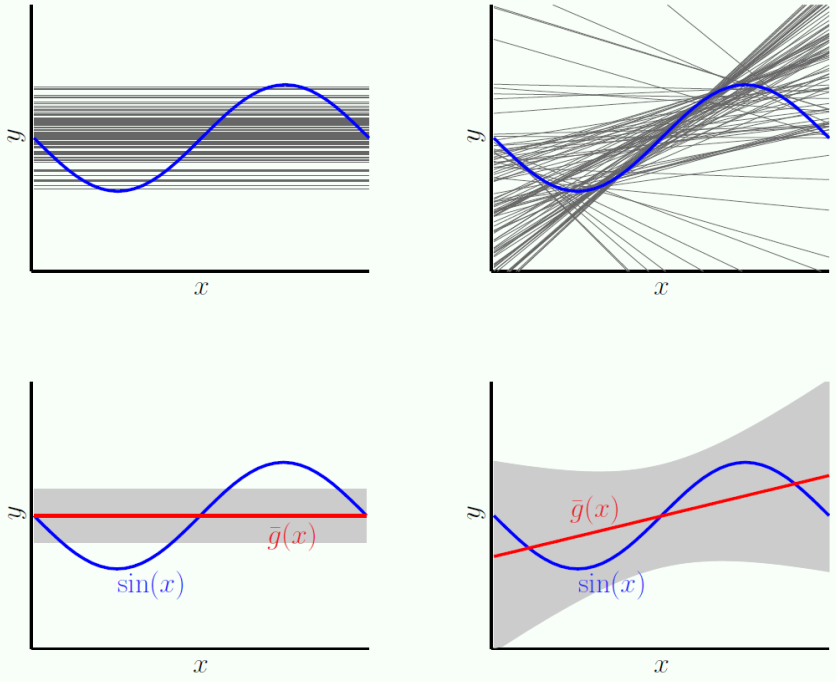
$\text{bias} = 0.21$

$\text{var} = 1.69$

$E_{\text{out}} = 1.90$

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

2 Data Points



$\mathcal{H}_0$

bias $= 0.50$;

var $= 0.25$.

$E_{\text{out}} = 0.75$ ✓

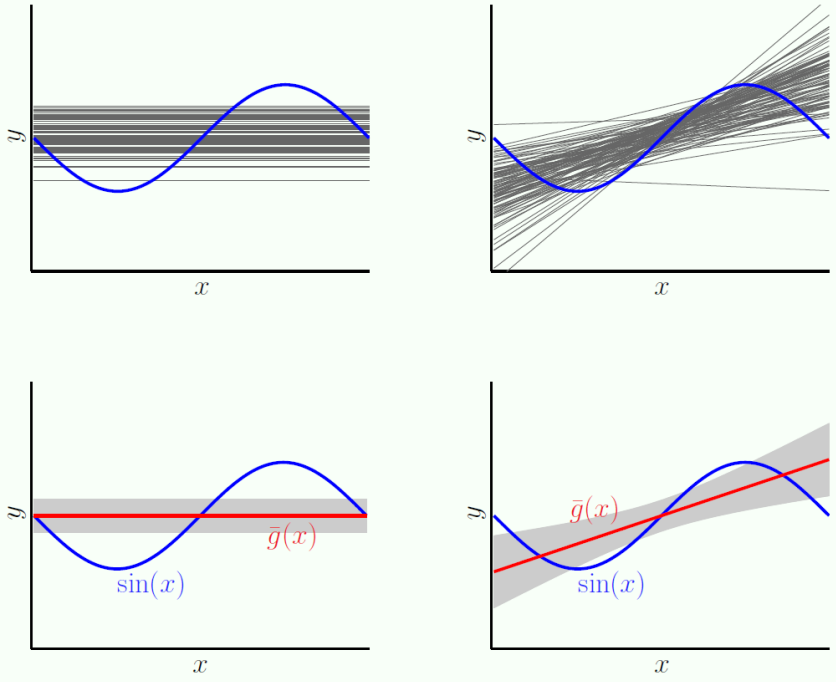
$\mathcal{H}_1$

bias $= 0.21$;

var $= 1.69$.

$E_{\text{out}} = 1.90$

5 Data Points



$\mathcal{H}_0$

bias $= 0.50$;

var $= 0.1$.

$E_{\text{out}} = 0.6$

$\mathcal{H}_1$

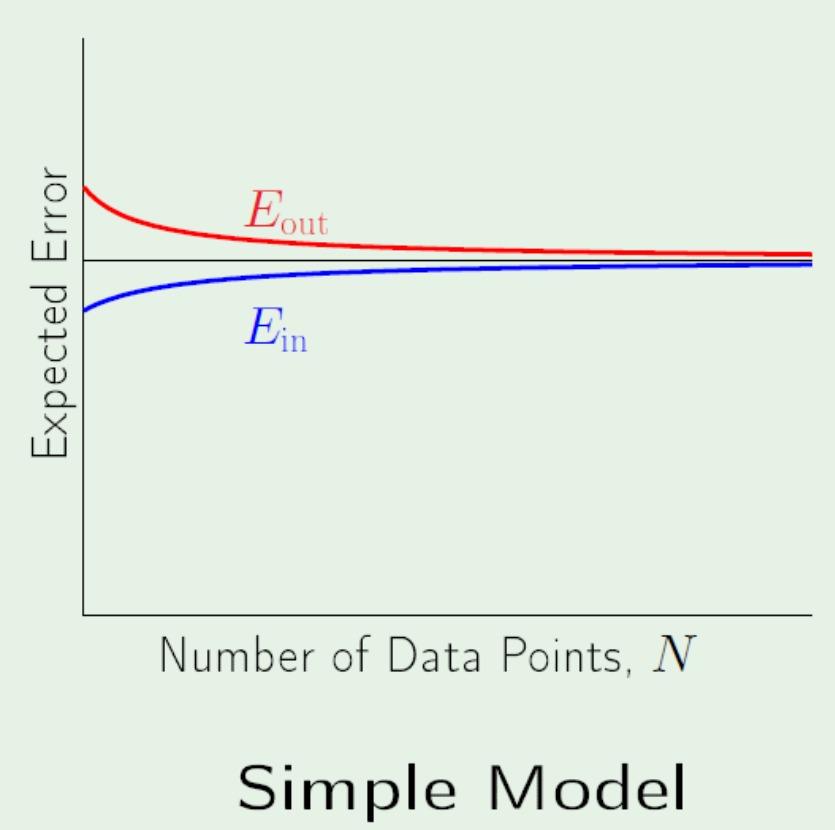
bias $= 0.21$;

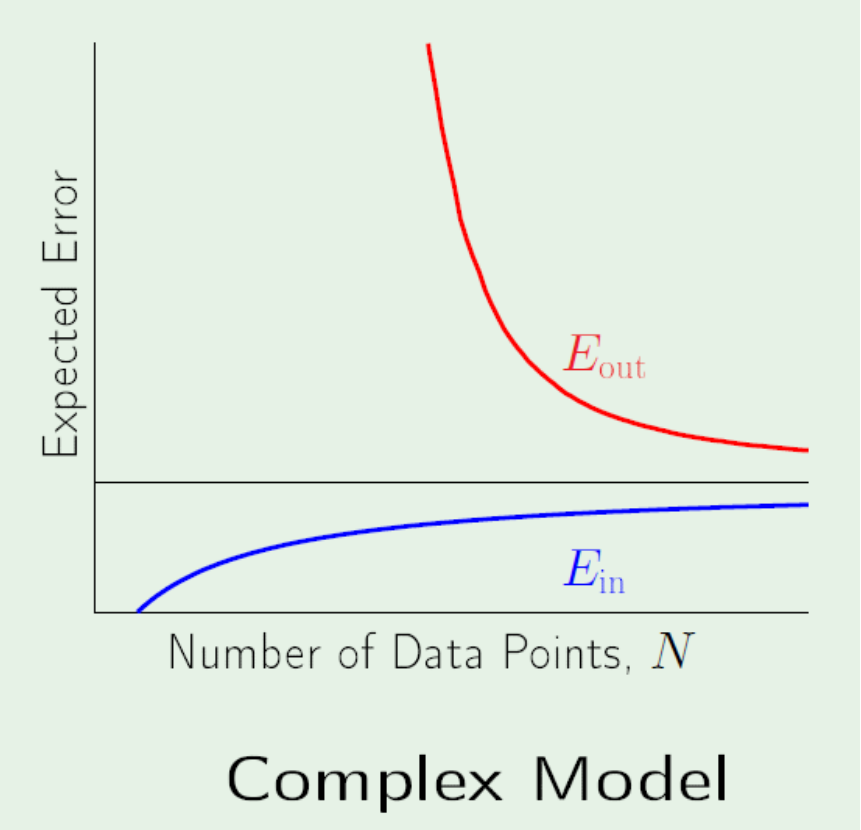
var $= 0.21$.

$E_{\text{out}} = 0.42$ ✓

Learning Curve

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





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

Left: 2nd order polynomial

Right. 10th order polynomial

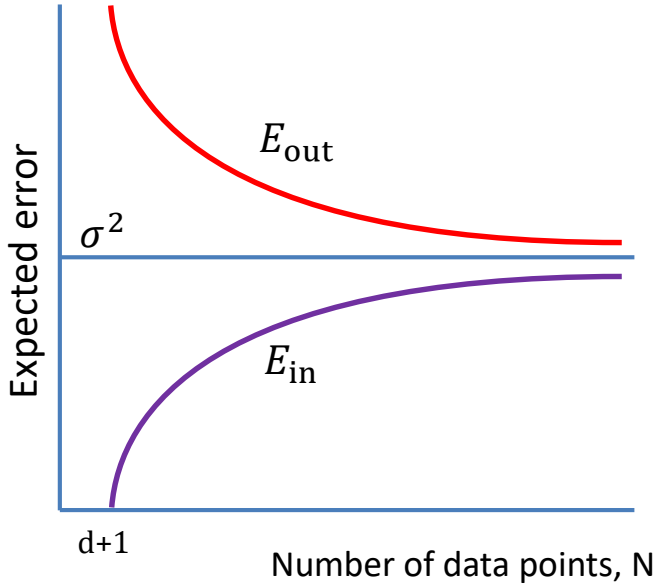
Learning Curve for Linear Regression

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

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

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

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



- E_{in} : When N increase the model absorbs as much information as possible with d+1 parameters
- E_{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 choosen

UNTIL NOW

Linear Models

From samples: $\{(\boldsymbol{x}_i, y_i), i = 1, \dots, N\}$

Classification: Perceptron

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

Error: $[\![sign(\boldsymbol{w}^T \boldsymbol{x}) \neq y]\!]$

Algorithms: PLA / Pocket

Regression Lineal:

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

Error: $(\boldsymbol{w}^T \boldsymbol{x} - \mathrm{y})^2$

Algorithms: SVD/SGD

Logistic Regression:

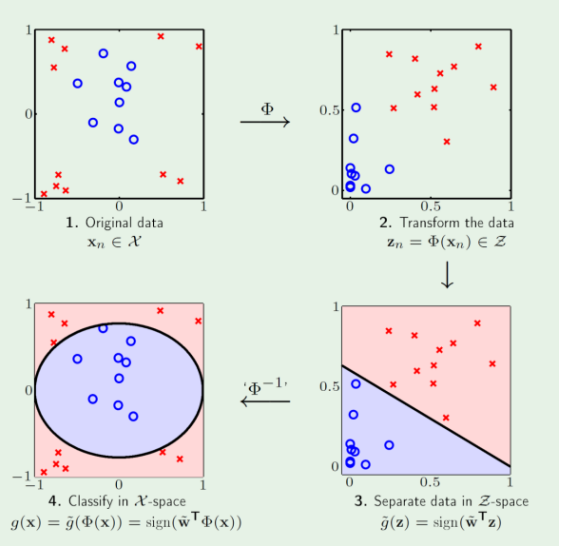
$h(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T \boldsymbol{x})$

Error: $1 + e^{-y\boldsymbol{w}^T \boldsymbol{x}}$

Algorithm: SGD

All of them use linear functions of the sampled features

Adding features



No-linear functions of the sampled features can be added to make $E_{in} \approx 0$

BUT which functions with $E_{in}(g) \approx 0$ also have $E_{out}(g) \approx 0$?

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+ = Sensitivity/(1-Specificity)
  - LR- = (1-Sensitivity)/Specificity

Pairs of Measures and Compounded Measures

- Precision / Recall
- Sensitivity / Specificity
- Likelihood Ratios (LR+ and LR-)
- Positive / Negative Predictive Values

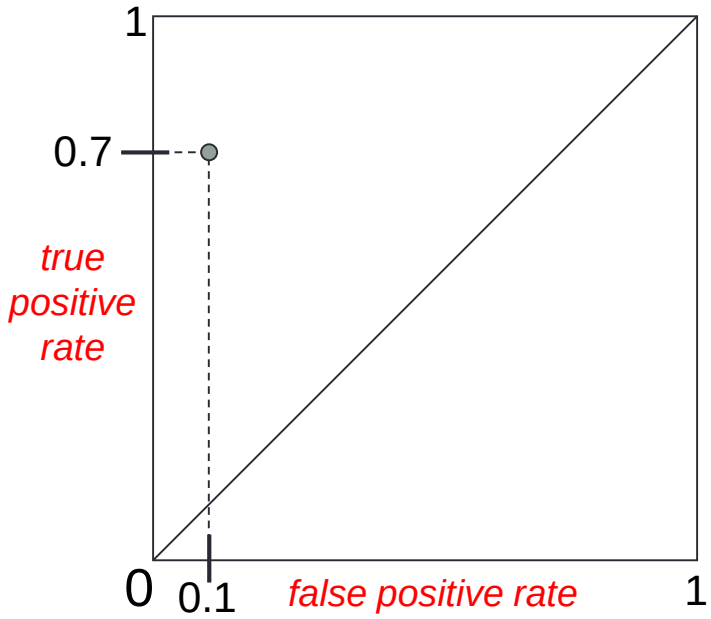
- F-Measure:
  - $F\alpha = [(1+ \alpha)\ (\text{Precision} \times \text{Recall})] / [(\alpha \times \text{Precision}) + \text{Recall}]$ $\alpha = 1, 2, 0.5$
- G-Mean: 2-class version single-class version
  - G-Mean = Sqrt(TPR x TNR) or Sqrt(TPR x Precision)

Evaluating the output (Binary case)

How can we measure the performance of a deterministic classifier?

"recall"

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



$$\frac{\text{\# false positives (FP)}}{\text{\# negatives (TN+FP)}}$$

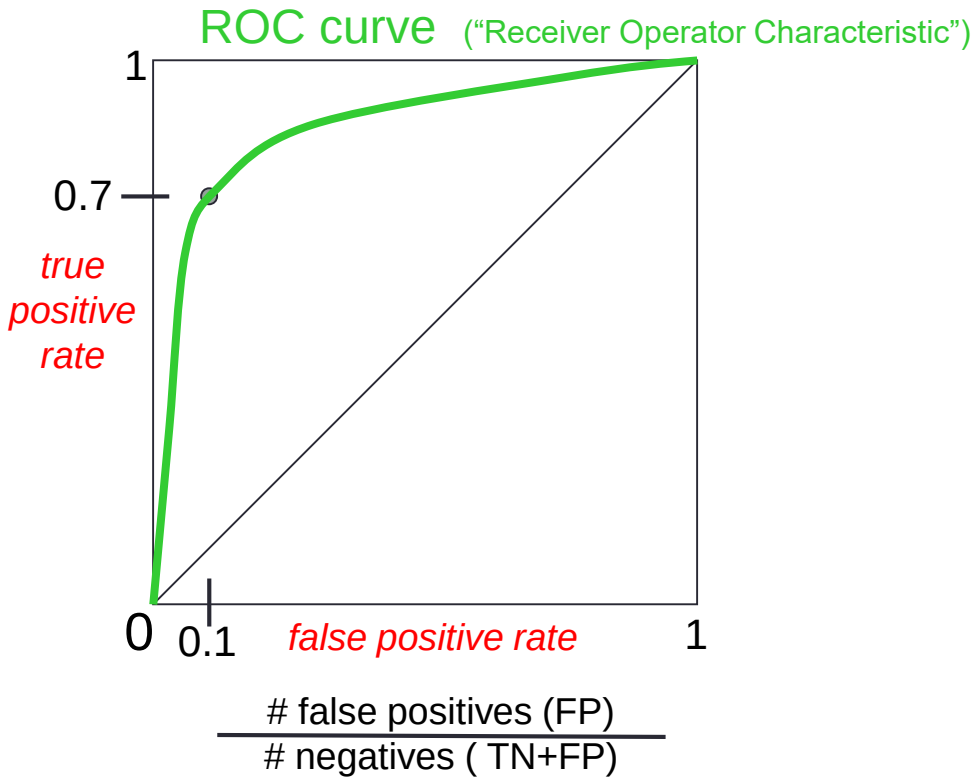
1 - "specificity"

ROC Curve (Binary Case)

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