

\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, $h(\mathbf{x}) = w_0 + w_1x_1 + \cdots + w_dx_d = w_0 + \sum_{i=1}^d x_iw_i$
 - 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 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: $h(\mathbf{x}) = \sum_{i=0}^d w_i x_i = \mathbf{w}^\top \mathbf{x}$

Elements of the problem

The data set: $(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \dots, (\mathbf{x}_N, \mathbf{y}_N)$

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

We've assumed $h_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1x_1 + \dots + w_dx_d = \mathbf{w}^T \mathbf{x}$

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

To measure the Error / Loss of $h_{\mathbf{w}}$ with respect to f , a quadratic error is used in regression.

$$\text{Error}(\mathbf{x}) = (h_{\mathbf{w}}(\mathbf{x}) - f(\mathbf{x}))^2$$

Then the sample error is

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

How to carry on with this ?

Linear Regression

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

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

- We want find \hat{h} such that, $\hat{h} = \min_{h \in \mathcal{H}} E_{out}(h)$
- Now \mathcal{H} takes the form: $h(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$
- In order to find \hat{h} we minimize the Empirical Risk (ERM):

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

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

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

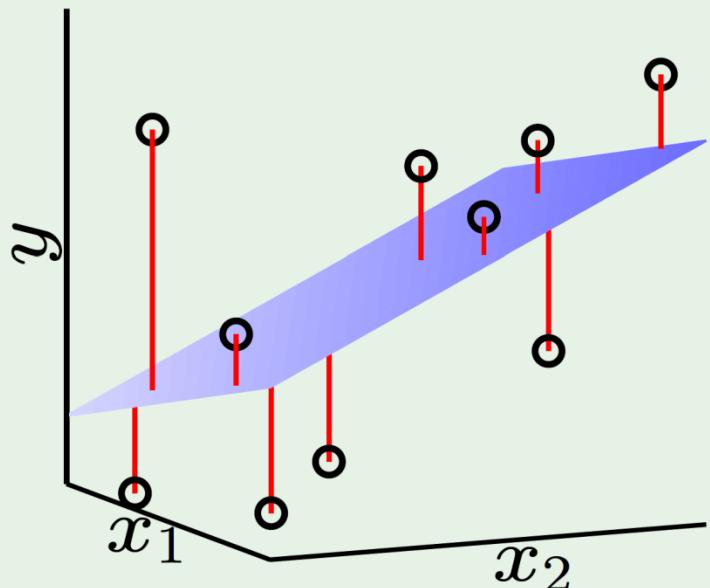
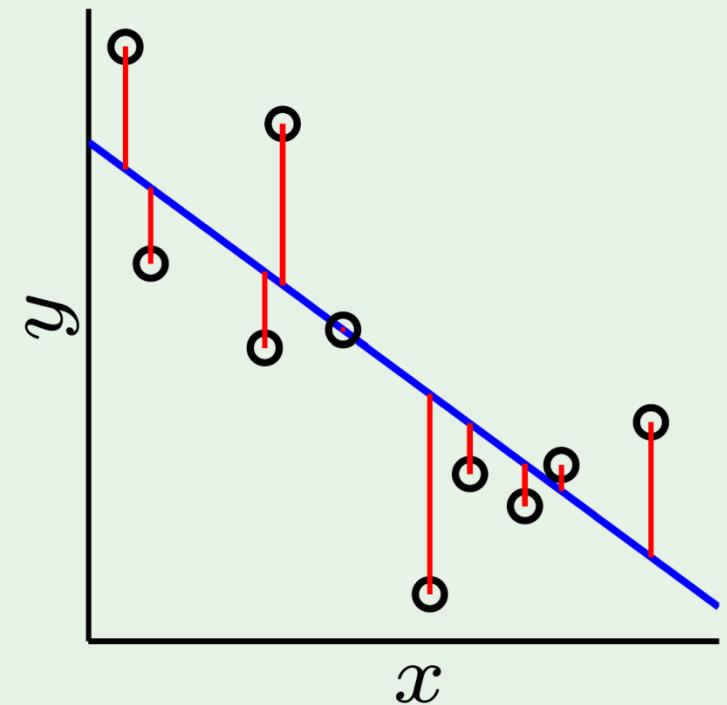
When is the quadratic error optimal?

- 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: $y_i = f(\mathbf{x}_i, \boldsymbol{\beta}) + \text{noise}$, f linear in $\boldsymbol{\beta}$
- **Theorem of Gauss–Markov:** Under the assumption of uncorrelated noise of zero mean and bounded variance, the OLS technique reaches the minimum variance unbiased estimator for $\boldsymbol{\beta}$

Equations:

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

Illustration of Linear Regression



A matrix expression for E_{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

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

Minimizing E_{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)** : $\mathbf{X} = \mathbf{U} \mathbf{D} \mathbf{V}^T$
 - $X^T X = V D D V^T$
 - $(X^T X)^{-1} = V^{-T} D^{-1} V^{-1} = V D^{-1} V^T$ 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

Some vector notation

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

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

- **HAT-MATRIX (\mathbf{H}):** Let \mathbf{X} ($N \times (d+1)$) denote the samples

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

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

$$\hat{\mathbf{H}} = \mathbf{X}(X^T X)^{-1} X^T \text{ map } \mathbf{y} \text{ to } \hat{\mathbf{y}}$$

- 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 dimension +1)
- **Generalization Error:** For linear regression exact formulas for the expected E_{out} and E_{in} can be found

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

- (see exercises in the book)

Alternative: Gradient Descent

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

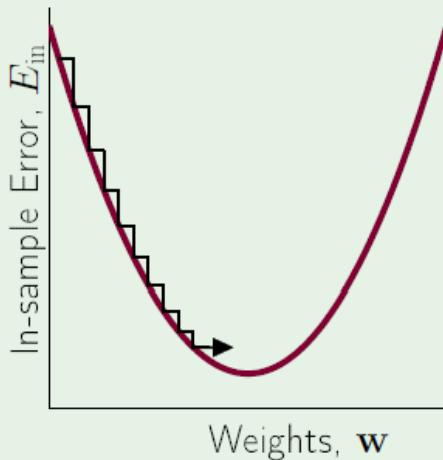
$$\begin{aligned}\Delta E_{in} &= E_{in}(\mathbf{w}(0) + \eta \hat{\mathbf{v}}) - E_{in}(\mathbf{w}(0)) \\ &= \eta \nabla E_{in}(\mathbf{w}(0))^T \hat{\mathbf{v}} + \mathcal{O}(\eta^2) \\ &\geq -\eta \|\nabla E_{in}(\mathbf{w}(0))\|\end{aligned}$$

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

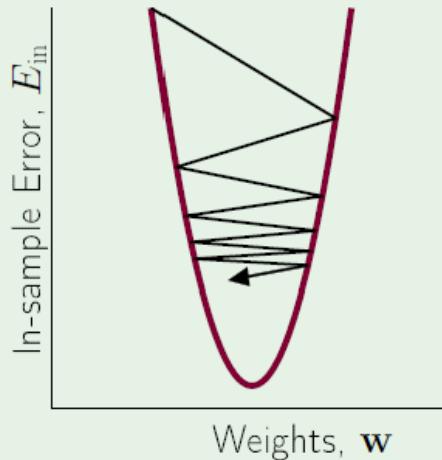
- Gradient Descent (GD):
 - Gradient descent is a general iterative optimization technique that reaches a local optimum following the direction of the gradient vector on each point.

How to fix η ?

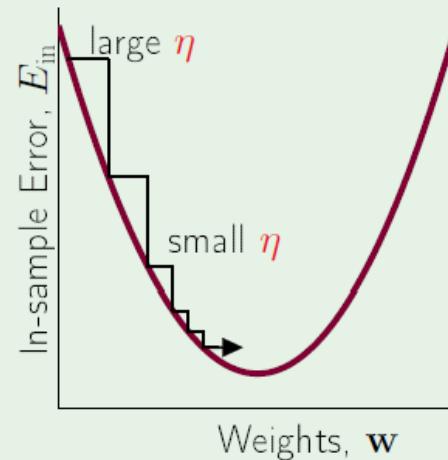
How η affects the algorithm:



η too small



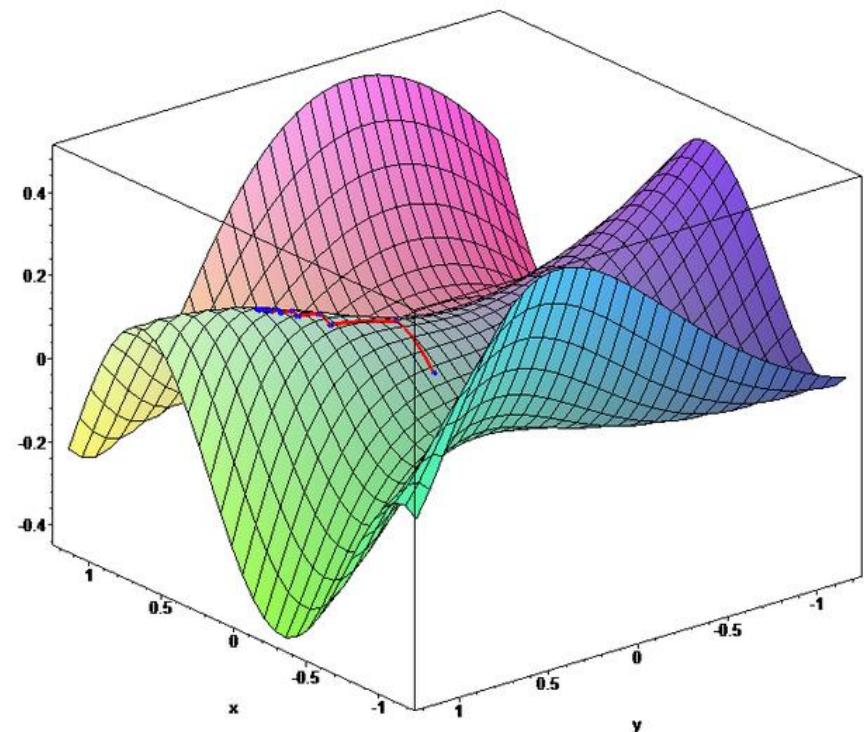
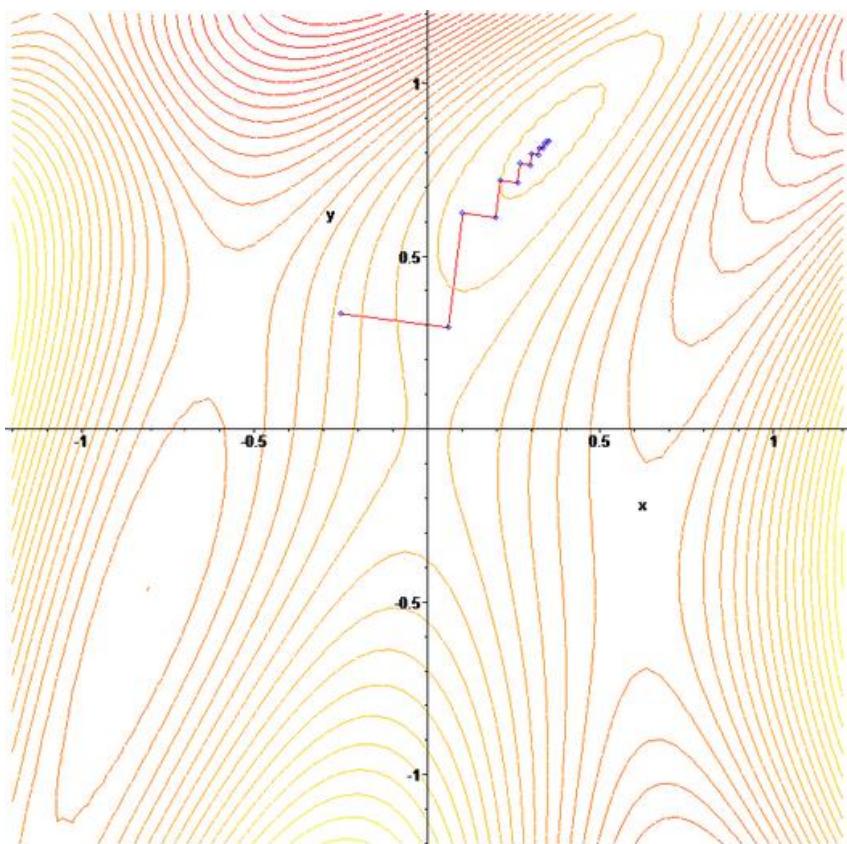
η too large



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 $j = 0, \dots, n$). Here, η is called the learning rate.

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

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**

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, $M \ll N$ (**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 (a rule of thumb is size: 32-256)

An iterative regression algorithm

Batch Gradient Descend

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

Stochastic Gradient Descend

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

Newton's Method: a cure for oscillation

- A new update rule for \mathbf{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.

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

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

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

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

The major complexity is to invert \mathbf{H}

Summary of Linear Regression

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

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

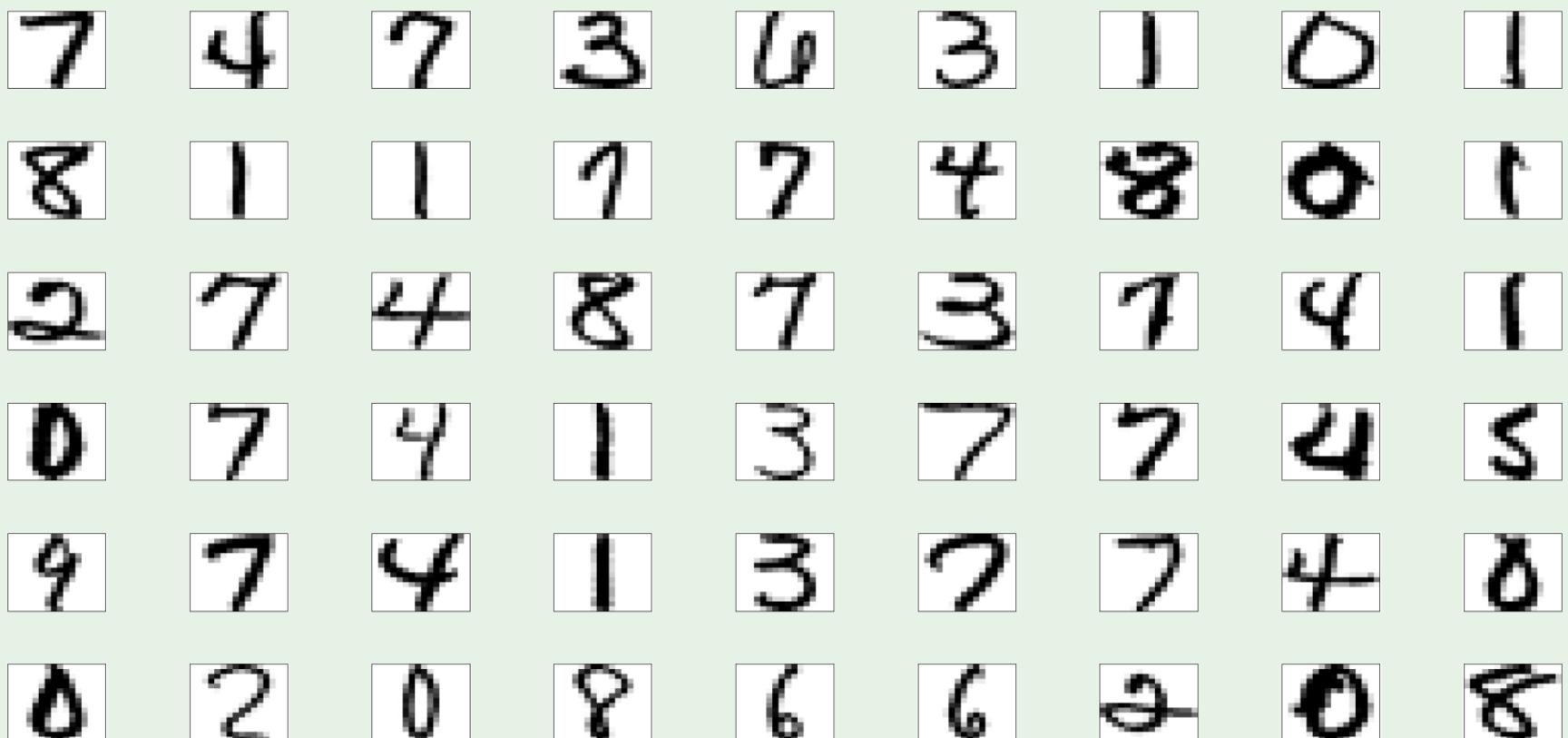
Categorical features & Lost values

- Categorical features encoding :
 - For lineal models a categorical feature is encoded as an one-hot vector: [0,0,0,...,1,...,0,0] which 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 $\mathbf{x} = (x_0, x_1, x_2, \dots, x_{256})$

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)

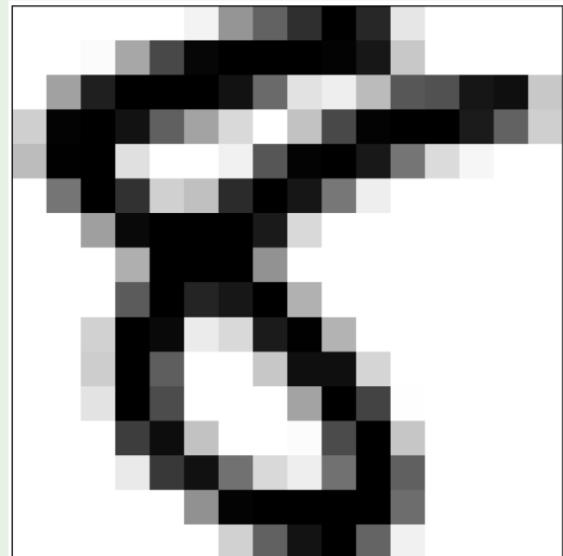
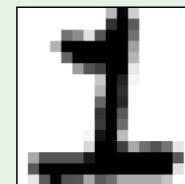
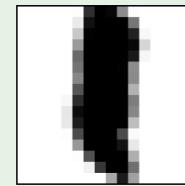
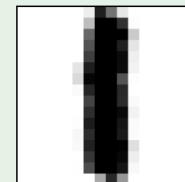
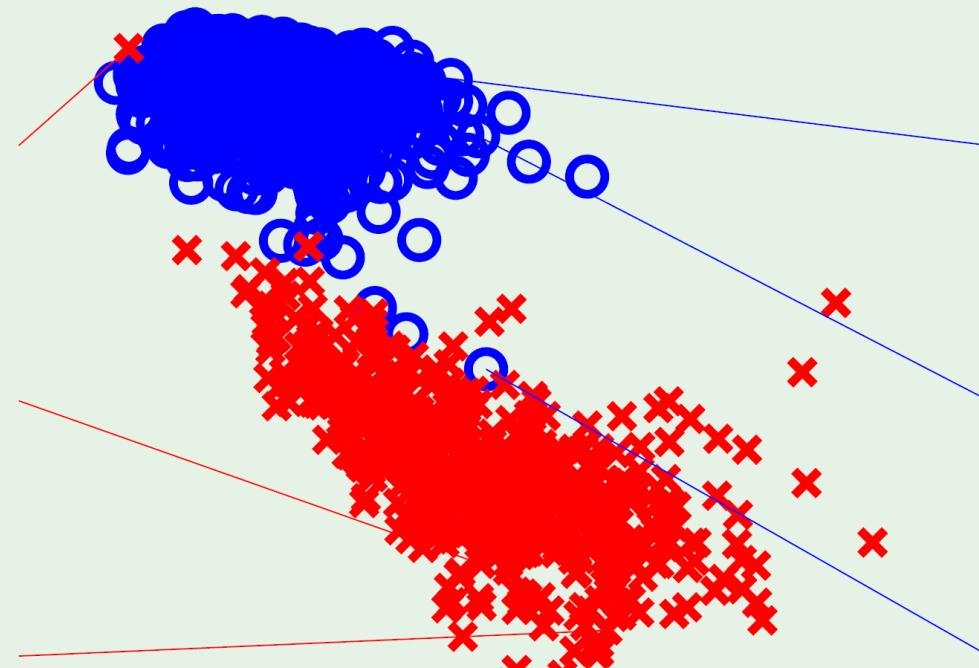
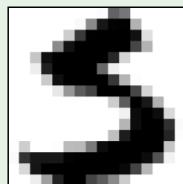
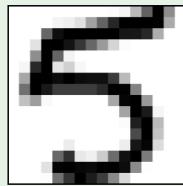
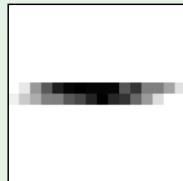


Illustration of features

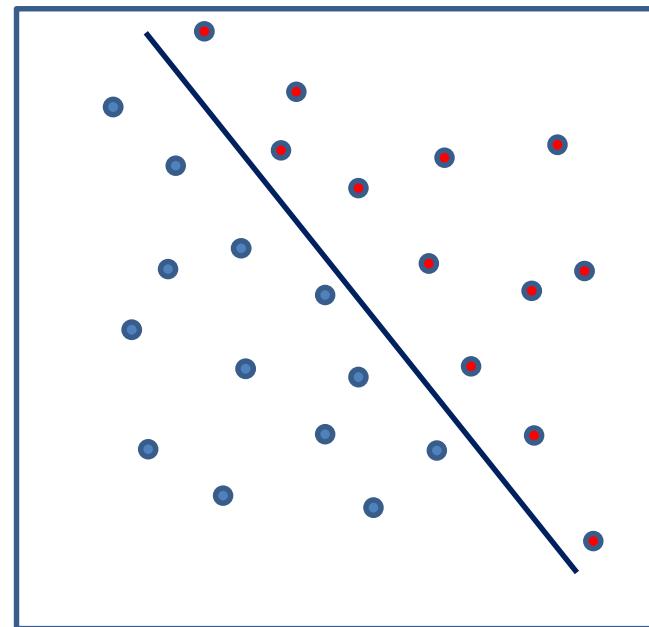
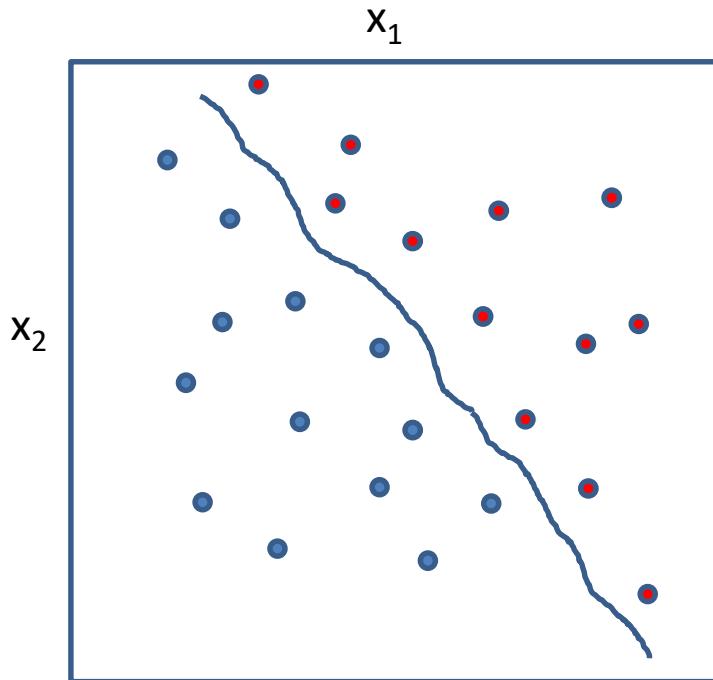
$$\mathbf{x} = (x_0, x_1, x_2)$$

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- Two classes

- Assignment 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 Deny credit

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

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

Perceptron or lineal separator

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

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

How to minimize it ?

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

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

Let's analyze in more detail

- The error function is

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

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

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

- Equivalently,

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

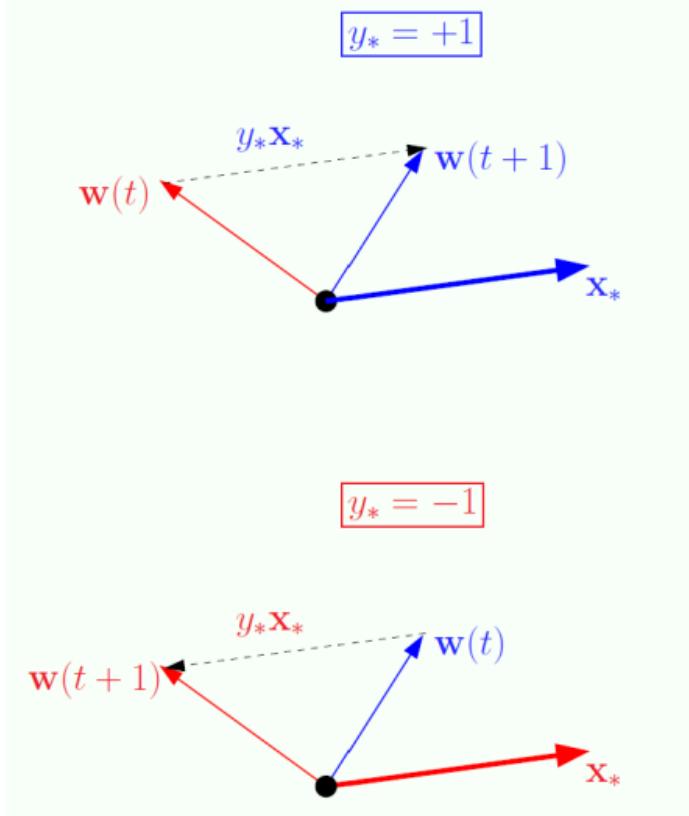
Perceptron Learning Algorithm (PLA)

Stochastic Gradient Descend with batch size =1 and learning rate $\lambda=1$

ADAPTING RULE :

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

- 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 \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 ! !

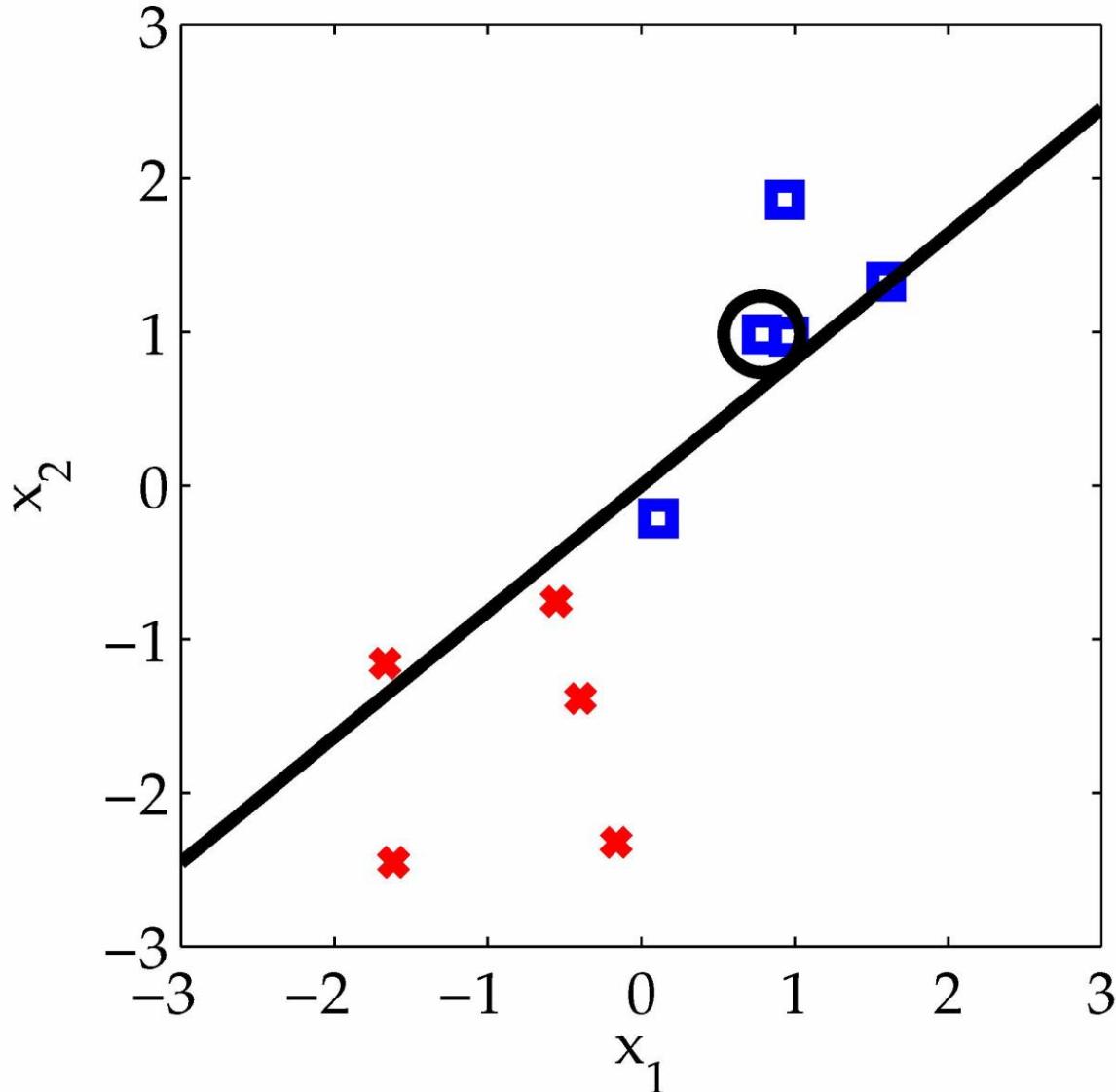


Linear Classification Algorithm

- 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}

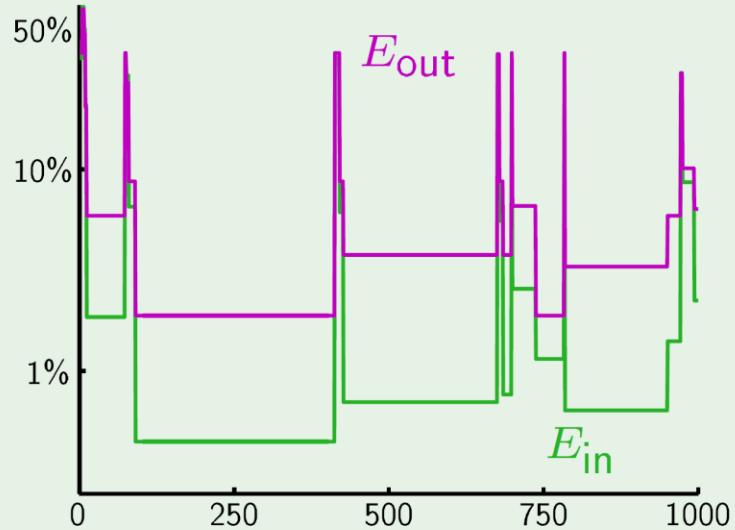
How the perceptron works

Iteration 1, Instance 1

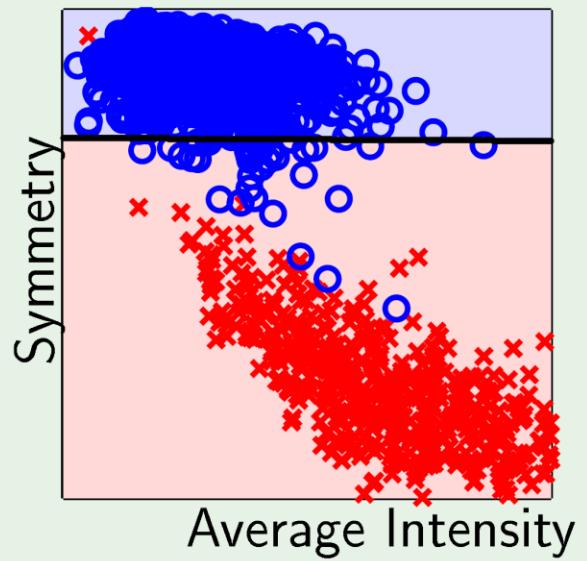


Error in PLA

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 $f: \mathcal{X} \rightarrow \mathcal{Y}$
- 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 $x_0 = 1$

- 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 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 $B = \min\{\|w\|\}: \forall i \in [m], y_i w^T x_i \geq 1, w \in R^d\}$ and let $R = \max_i \|x_i\|$. Then the PLA stops after at most $(RB)^2$ iterations, and when it stops it holds that $\forall i \in [m], y_i w^T(t) x_i \geq 0$.
- 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 scenarios:

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

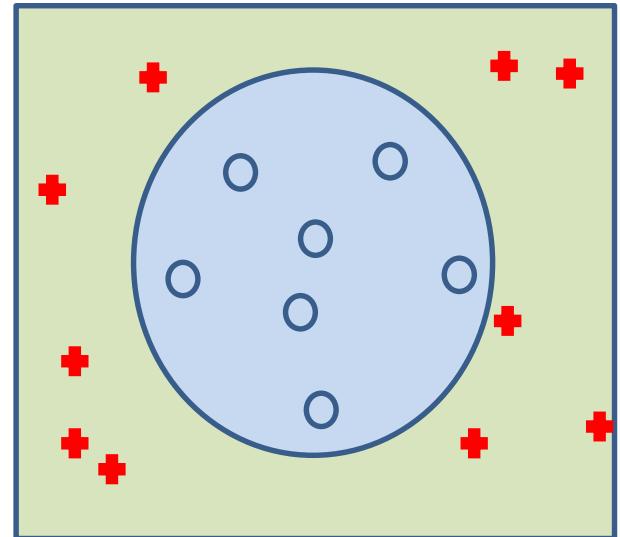
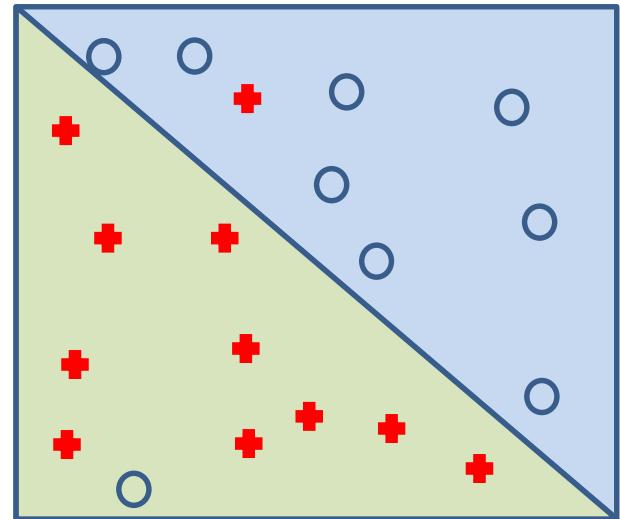
To find a solution in the NS.1 we have to solve

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

But this problem is known to be a 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)

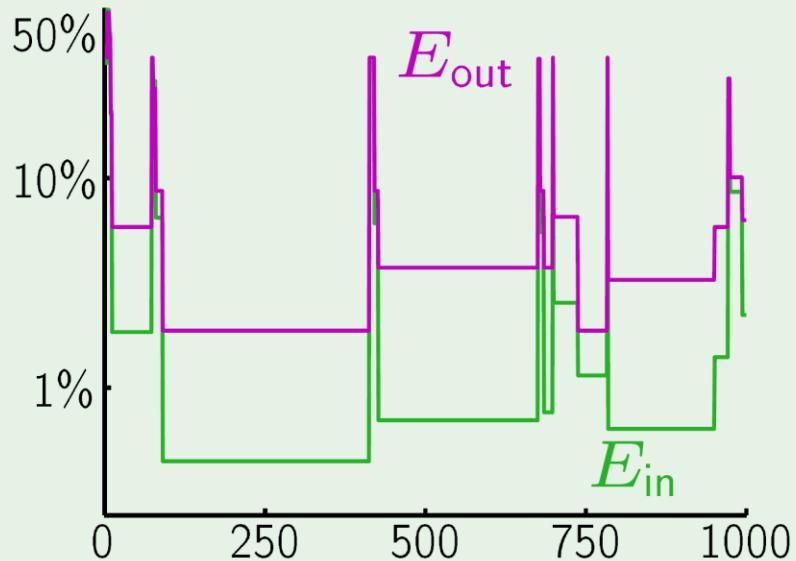


Non-Separable Case: noisy labels

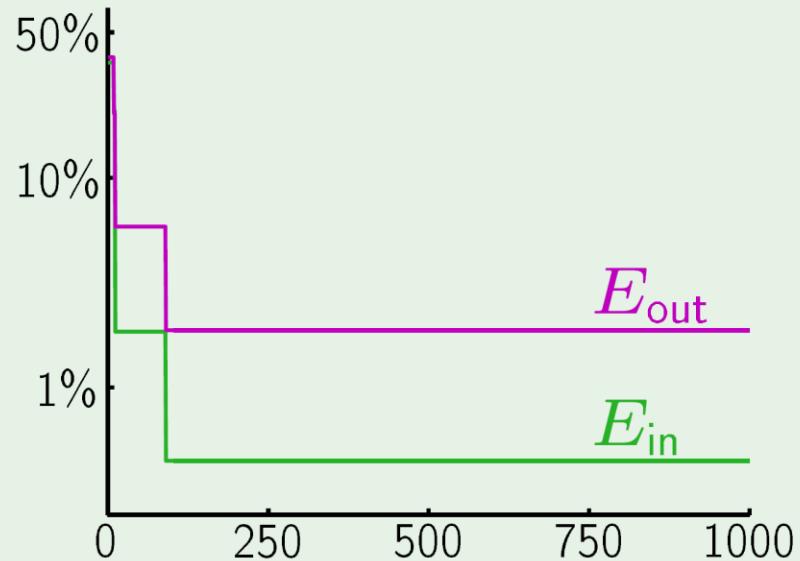
- The Pocket Algorithm: essentially the pocket algorithm keeps “in its pocket” the best vector solution encountered up the iteration t in PLA
- POCKET ALGORITHM:
 1. Set the pocket weight vector \hat{w} to $w(0)$ of PLA
 2. **for** $t=1, \dots, T$ **do**
 3. Run PLA for one update to obtain $w(t+1)$
 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:

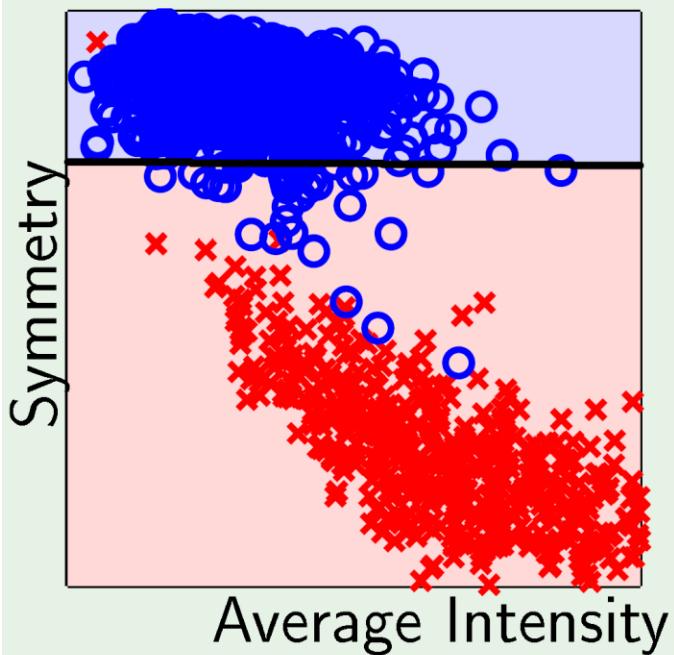


Pocket:

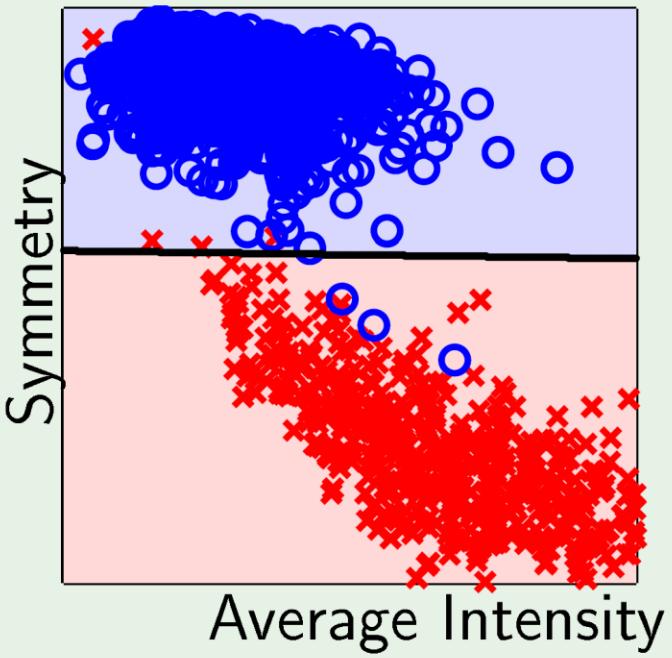


Classification boundary – The PLA vs Pocket

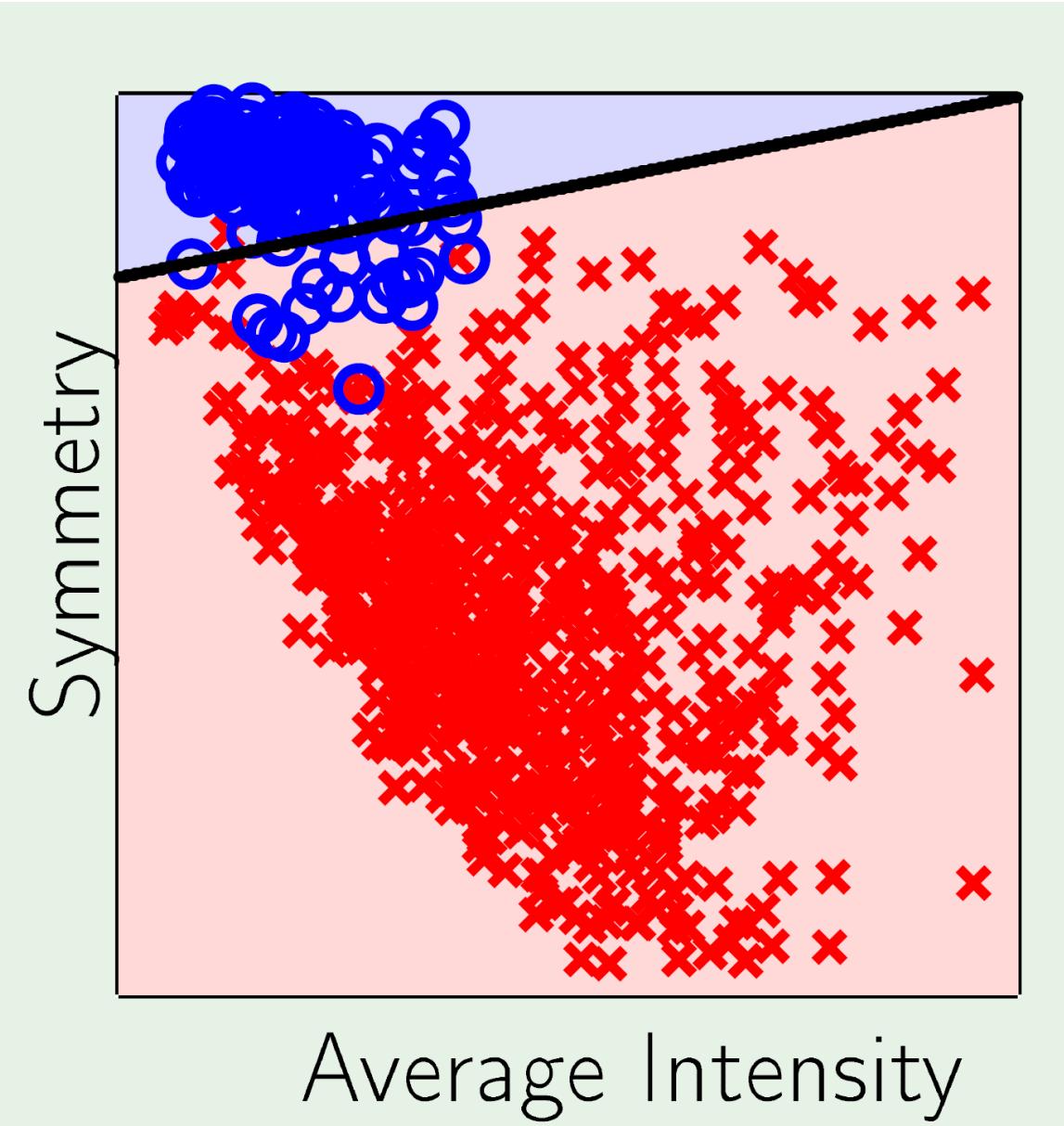
PLA:



Pocket:



Linear Regression Boundary

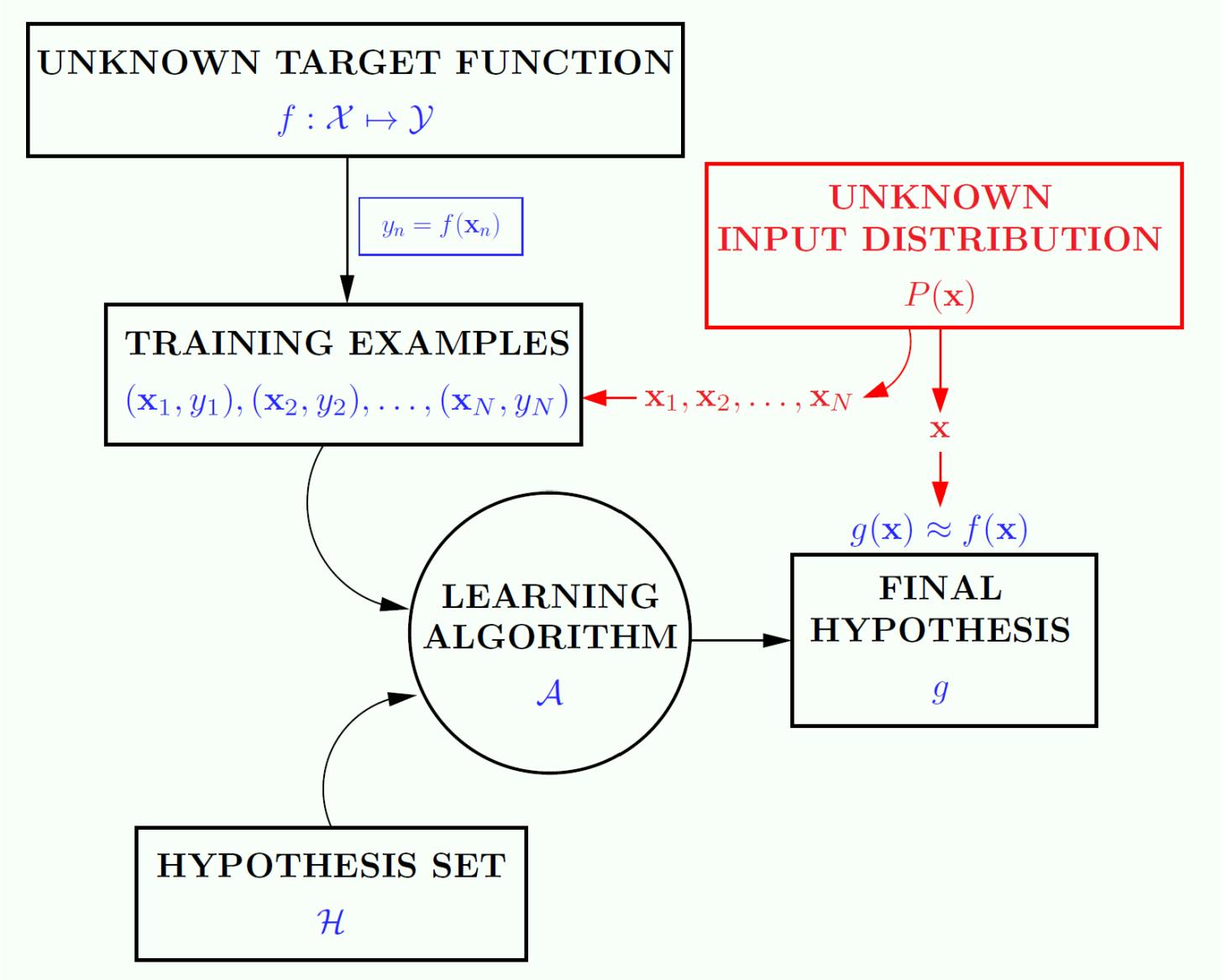


Linear Regression for Classification

- 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 !

Noisy labels: A general setup

Current Learning Setup



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 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

$$P(y|x)$$

- 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

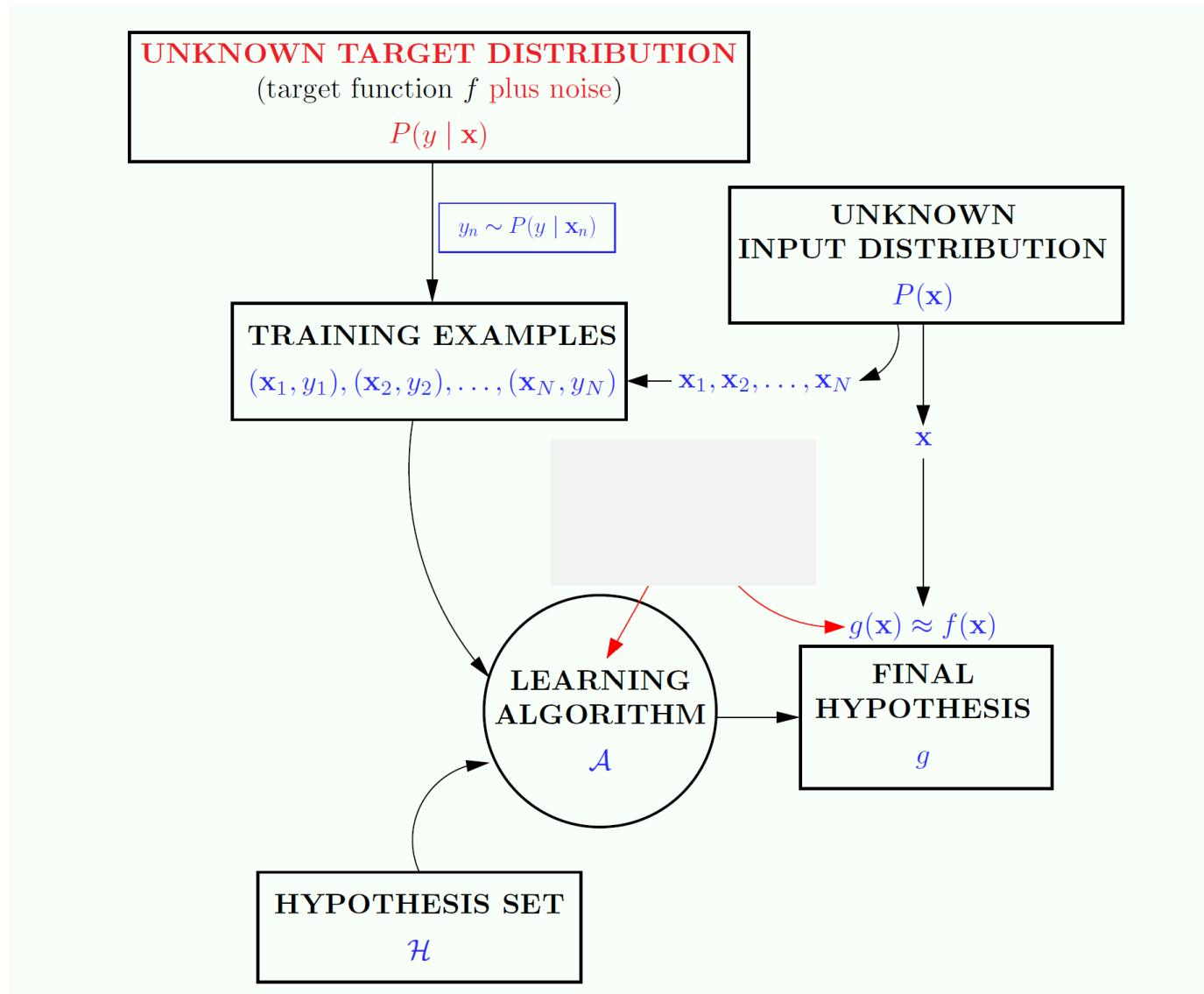
$P(y|x)$ is zero except in $y = f(x)$

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)

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

Updated Learning Setup



PROBABILITY ESTIMATION

Logistic Regression (LGR)

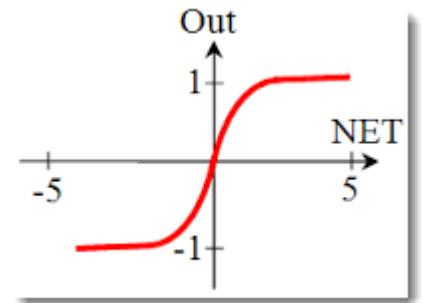
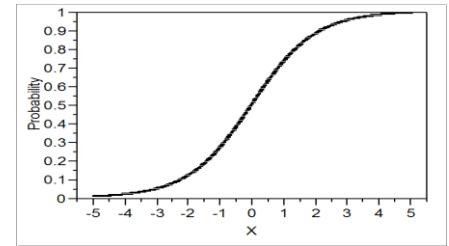
- Linear Regression: $h(x) = \mathbf{w}^T \mathbf{x}$,
- **Logistic Regression:** $h(x) = \sigma(\mathbf{w}^T \mathbf{x}) \in [0,1]$ or also $\tanh \in [-1,1]$

- The σ -function is called *logistic function (sigmoide)*:

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

- Other function:

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



- 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:

- $h(\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x}) = \frac{e^{\mathbf{w}^T \mathbf{x}}}{1+e^{\mathbf{w}^T \mathbf{x}}}$ C1 probability
- $1 - h(\mathbf{x}) = \sigma(-\mathbf{w}^T \mathbf{x}) = \frac{1}{1+e^{\mathbf{w}^T \mathbf{x}}}$ C2 probability

The ratio $\frac{h(\mathbf{x})}{1-h(\mathbf{x})}$ is called “odds”

- $\ln(\text{odds}) = \ln\left(\frac{h(\mathbf{x})}{1-h(\mathbf{x})}\right) = \ln\left(e^{\mathbf{w}^T \mathbf{x}}\right) = \mathbf{w}^T \mathbf{x}$ (función logit)
- **The logit function can be any regression model**

Logistic Regression (LGR)

- Now the unknown labeling function is

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

- But the label values we are given are deterministic, (± 1) – labels.
- As the (± 1) – labels are highly correlated with the function f , we choose as $h(\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x})$ the \mathbf{w} -vector that maximizes the probability of the simple $\mathbf{s} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$ given by

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

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

Learning Criteria: ML

- 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 a hypothesis h , how close is h 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 sample points are independents, the probability of the sample is

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

Error Measure (Loss)

Maximum Likelihood (ML): 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 $y_i \mathbf{w}^T \mathbf{x}_i \gg 0$

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

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
- Let x be a single Bernoulli random variable : $P(x=1)=p$ and $P(x=0)=q$, $p+q=1$
- Probability of any value of x : $P(x)=p^x q^y$, $y=1-x$, $x \in \{0,1\}$
- $P(x_1, x_2, \dots, x_N) = P(x_1) \times P(x_2) \times \dots \times P(x_N) = p^{\sum x_i} q^{\sum y_i} = p^{\# [x=1]} q^{\# [x=0]}$
- $L(x_1, x_2, \dots, x_N) = \prod_{i=1}^N \sigma(\mathbf{w}^T \mathbf{x}_i)^{[x_i==1]} (1 - \sigma(\mathbf{w}^T \mathbf{x}_i))^{[x_i==0]} = \prod_{i=1}^N \prod_{k=0}^1 \sigma(\mathbf{w}^T \mathbf{x}_i)^{y_{ik}}$
- 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 \mathbf{y} are represented by **1-of-K vector**: $[0,0,\dots,1,\dots,0]$ or $[-1,-1,\dots,1,\dots,-1]$

The sample likelihood (K clases)

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

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

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

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

Multilabel classification: SoftMax

- $t_{nk} = \sigma(\mathbf{w}_k^T \mathbf{x}_n)$, $\sum_{k=1}^K t_{nk} = 1$,
- $E(\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_K) = -\ln L(\mathbf{Y} | \mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_K) = -\sum_{n=1}^N \sum_{k=1}^K y_{nk} \ln t_{nk}$
- $\nabla_{\mathbf{w}_j} E(\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_K) = \sum_{n=1}^N (t_{nj} - y_{nj}) \mathbf{x}_n$
- **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 $P(C_j | \mathbf{x}) = \max_k P(C_k | \mathbf{x})$, $k = 1, \dots, K$

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).

ERM: Learning Rule

- The Goal: Minimize the sampling error to choose a final hypothesis with low E_{out} error.
- **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: $ERM_{\mathcal{H}}(\mathcal{D}) = \operatorname{argmin}_h \left\{ \frac{1}{N} \sum_{i=1}^N (y_i - h(\mathbf{x}_i))^2 \right\}$
 - Logistic regression: $ERM_{\mathcal{H}}(\mathcal{D}) = \operatorname{argmin}_h \left\{ \frac{1}{N} \sum_{i=1}^N \ln(1 + e^{-y_i h(\mathbf{x}_i)}) \right\}$
- 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 $P(Y | X)$

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

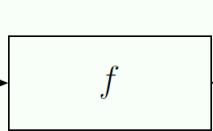
- When $f \in \mathcal{H}$, is expected that $ERM_{\mathcal{H}}(\mathcal{D}) \leq \varepsilon$ for some finite N-value

SGD+ERM: A new induction rule

- We can apply SGD to solve any optimization problem where the gradient can be computed.
- Example.1 : Regression
 - The error function is $e_n(\mathbf{w}) = (y_n - \mathbf{w}^T \mathbf{x}_n)^2$
 - 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 : 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.2 : 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

Cost of error

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	false accept
	-1	false reject	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	10	0

Supermarket

		f	
		+1	-1
h	+1	0	1000
	-1	1	0

CIA

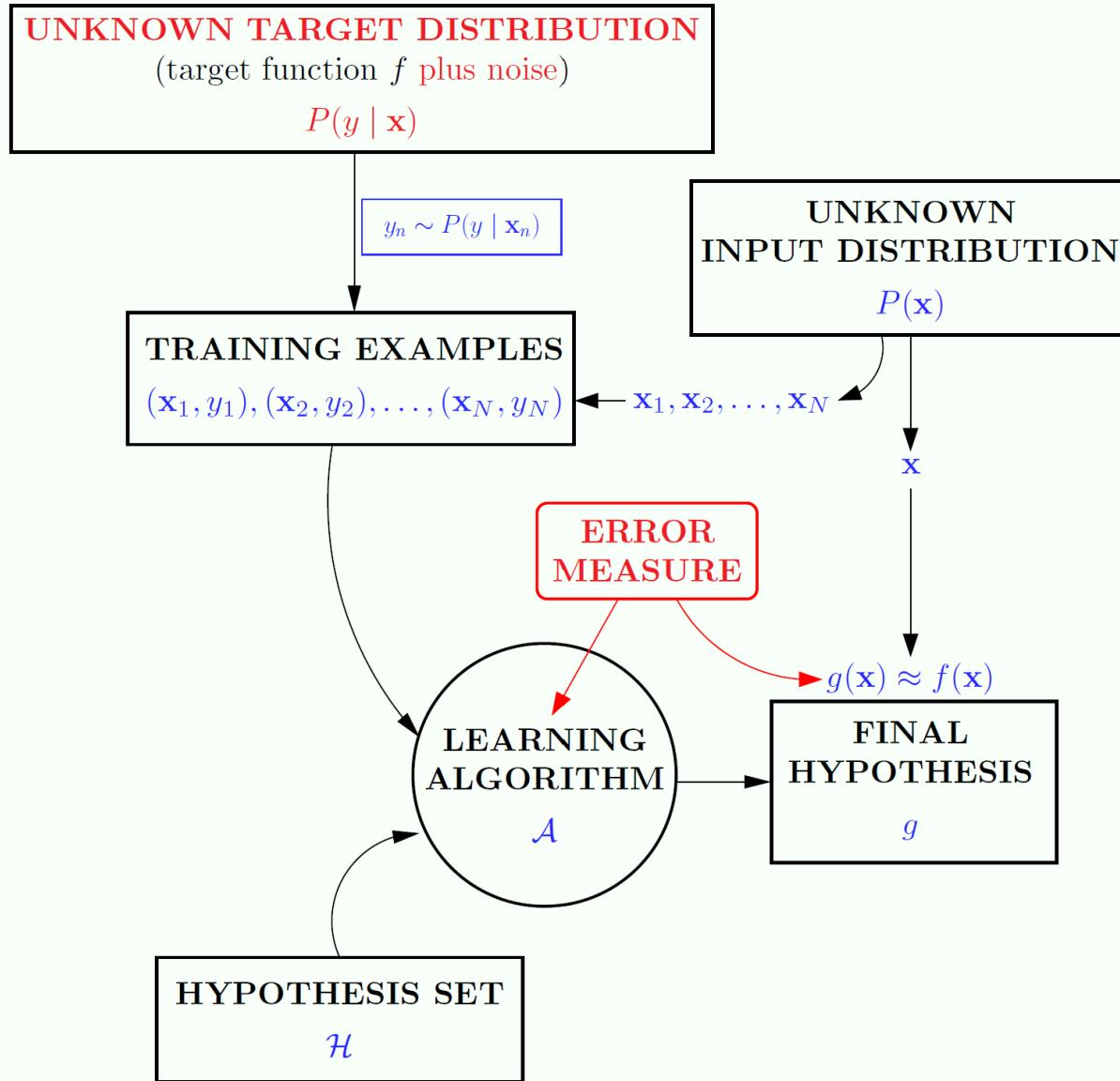
Take Away

Error measure is specified by the user.

If not, choose one that is

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

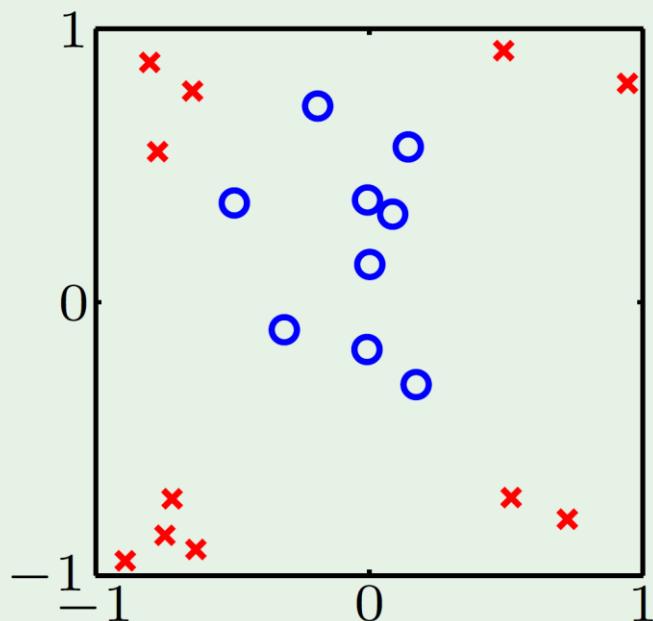
Learning Setup with Error Measure and Noisy Target



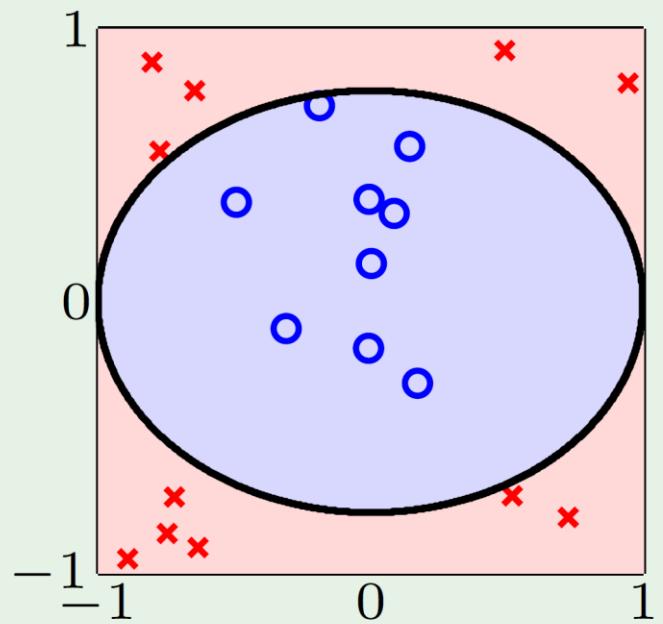
Adding functions to \mathcal{H} : Non linear Transformations

Linear predictors are limited

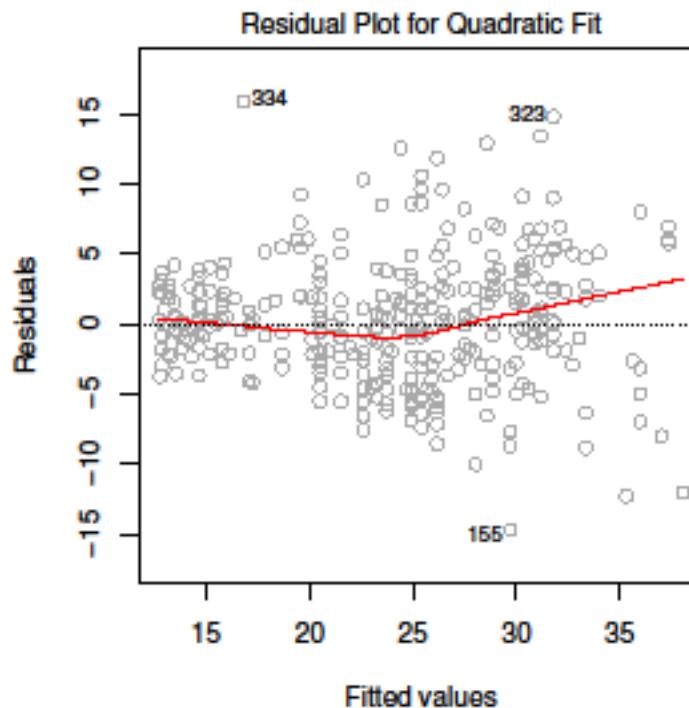
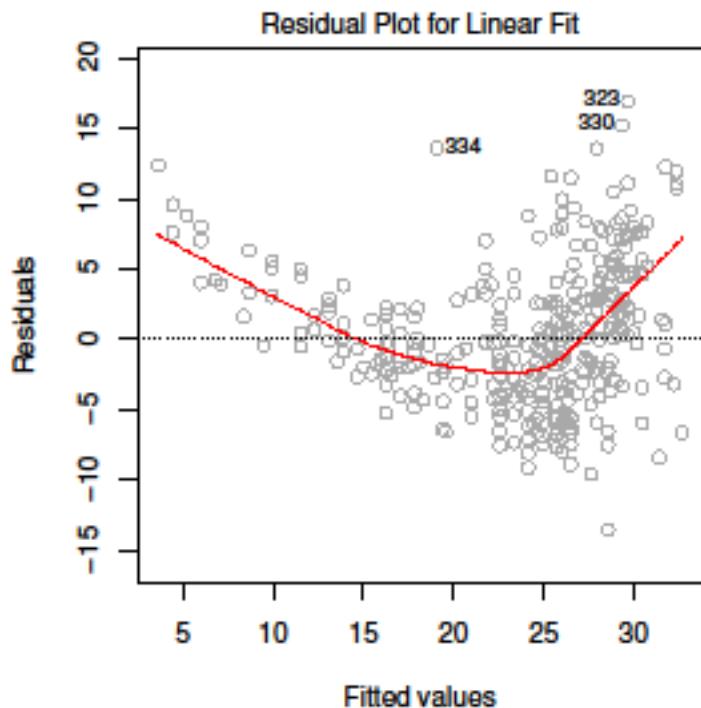
Data:



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 features: $\llbracket x_i > 5 \rrbracket$ and $\llbracket x_i < 1 \rrbracket$ could be better features

How can we do this with linear models ?

- Linear regression computes:

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

- Linear classification computes:

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

Algorithms work because of 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 features 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 !!

Polynomial Regression

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

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

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

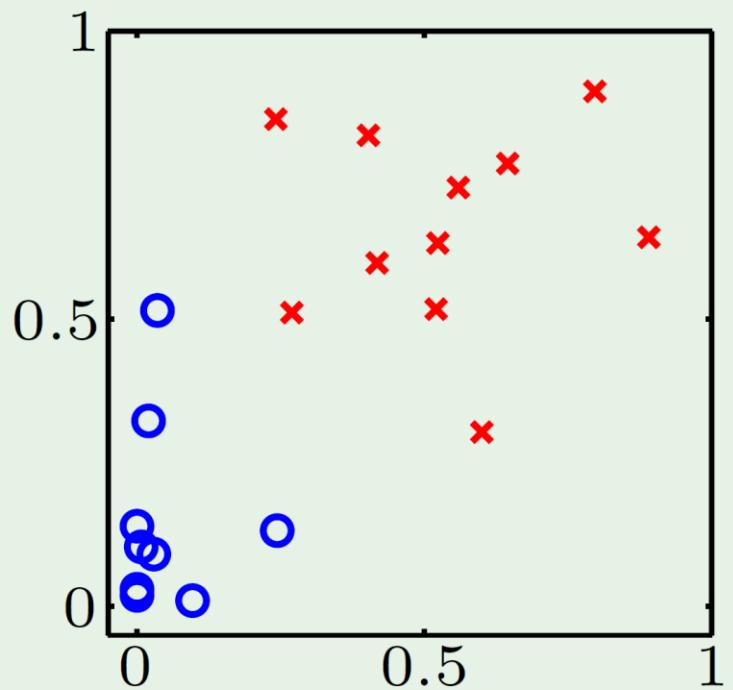
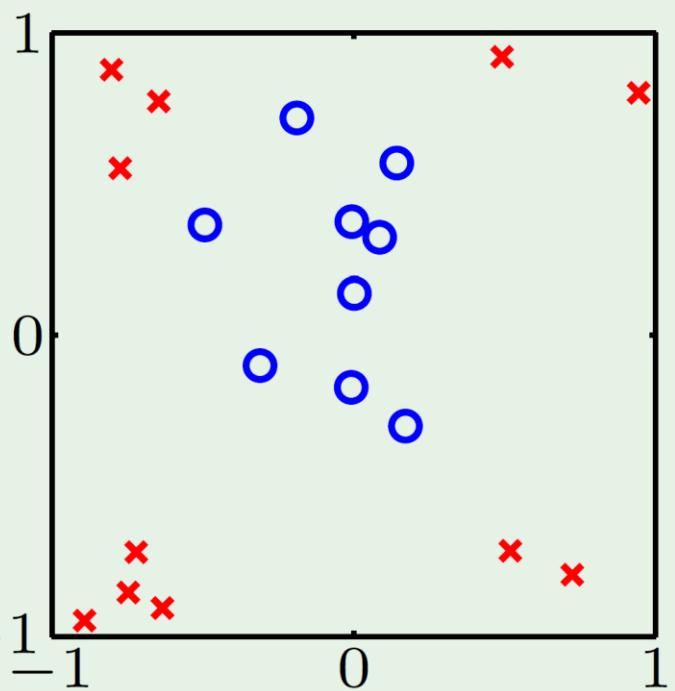
- Clearly all predictors are independent
- The matrix X for sample data of size N is defined as

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

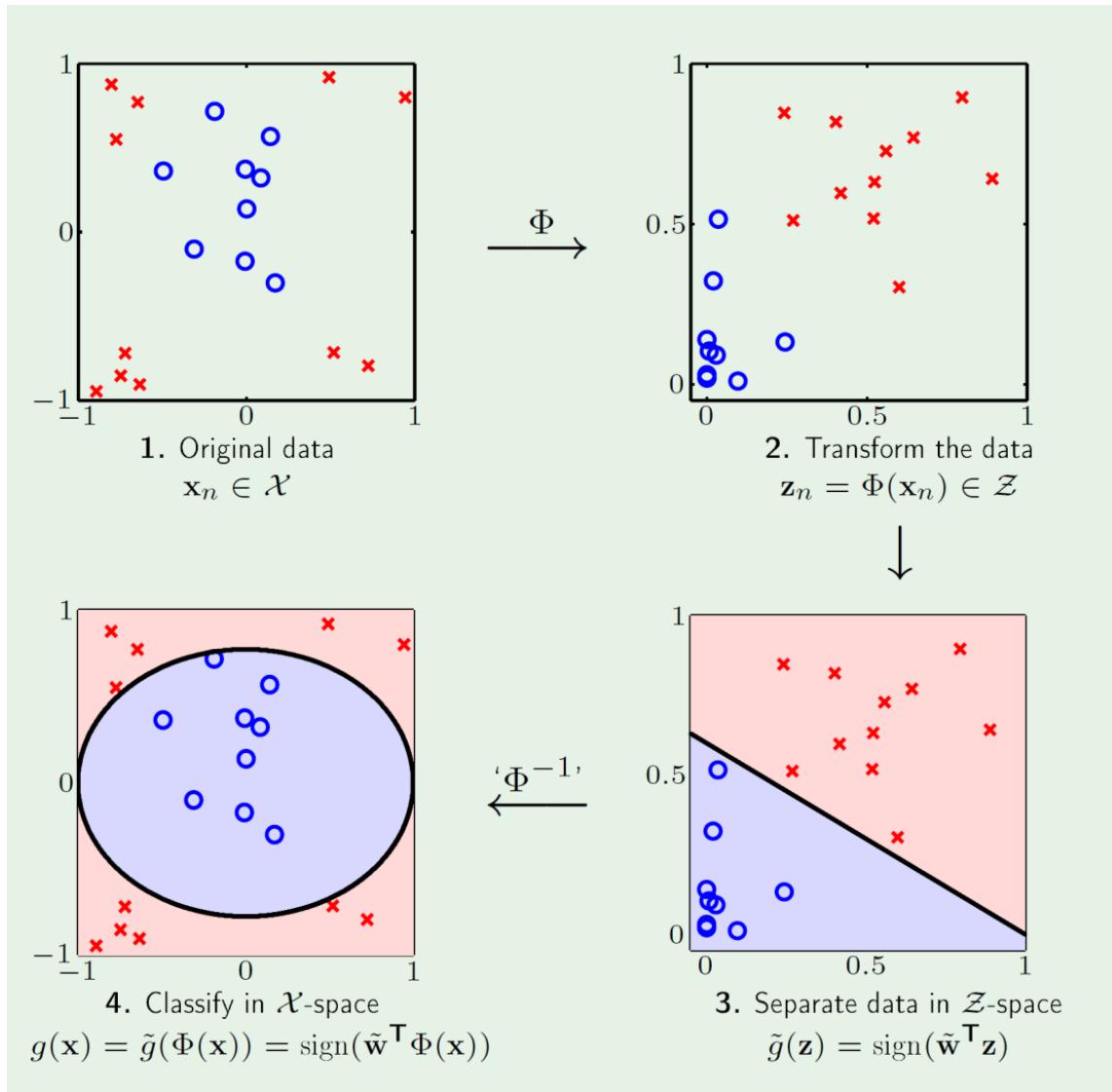
- How many terms to choose ? We talk on it later on.

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}

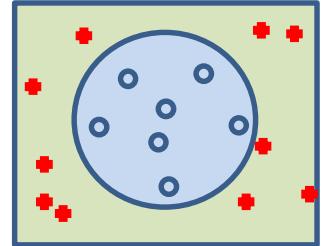
$$\tilde{\mathbf{w}} = (w_0, w_1, \dots, w_{\tilde{d}})$$

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

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

$$(x_1, x_2, 1) \xrightarrow{\Phi} (x_1^2, x_2^2, 1) = (\mathbf{z}_1, \mathbf{z}_2, 1)$$

- Let write the solution using \mathbf{z} : $\tilde{h}(\mathbf{z}) = \text{sign}((1)\mathbf{z}_1 + (1)\mathbf{z}_2 - 0.6)$ Fine!!
$$\tilde{h}(\mathbf{z}) = \tilde{h}(\Phi(\mathbf{x})) = h(\mathbf{x})$$
- In real problems, we must set the transform before seeing the data. So, we do not know in advance the better specific transform.

Generalized Error/Loss Functions

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

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

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

0-1 loss

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

Square loss

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

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

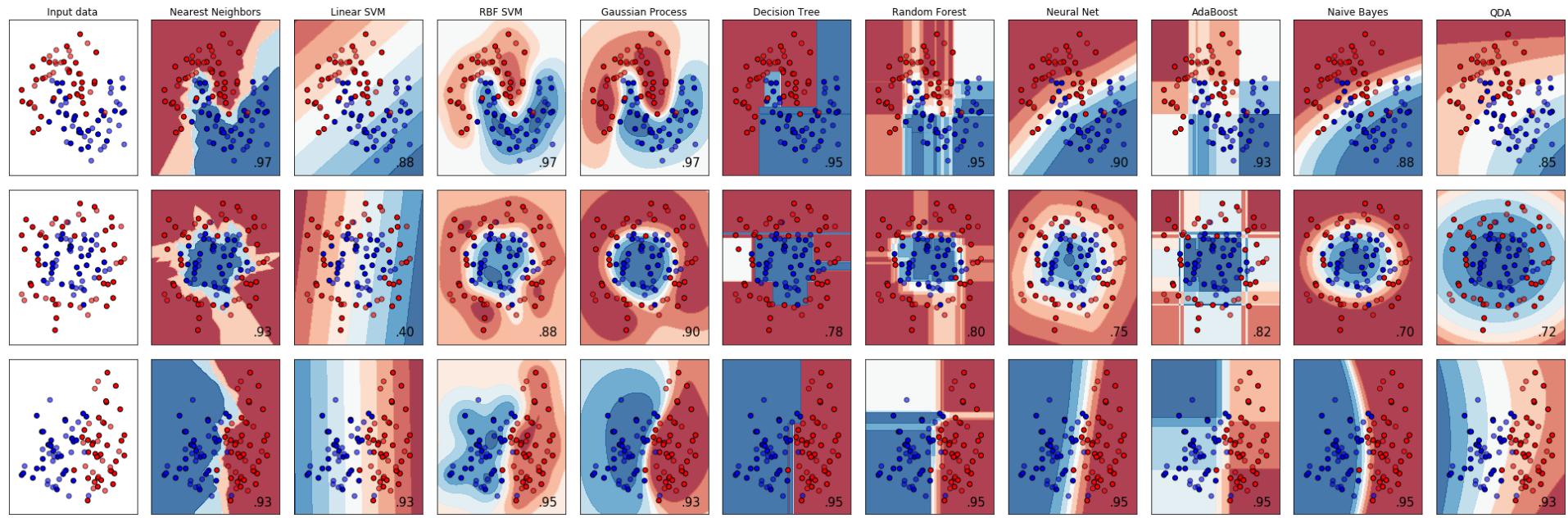
- That definition is an average on all possible examples ($\text{risk}(h) = \text{average loss}$)
- This expresion is applicable in any learning setting

Computation and Generalization

- Let denote by Φ_Q the *Q-th order polynomial transform*
 - $\Phi_4(x) = (1, x_1, x_2, x_1^2, x_2^2, x_1x_2, x_1^3, x_2^3, x_1x_2^2, x_2x_1^2, x_1^4, x_2^4, x_1^2x_2^2, x_2^1x_1^3, x_1^1x_2^3)$
- 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 $d = \frac{Q(Q+3)}{2} = \mathcal{O}(Q^2)$ 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. (remember $\mathcal{O}(dN^{-1}\log(N))$)
- In general when choosing the appropriate dimension for the feature transform, we must use an approximation-generalization tradeoff:

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

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



2D intuition is lost in high dimensionality

EVALUATION METRICS

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

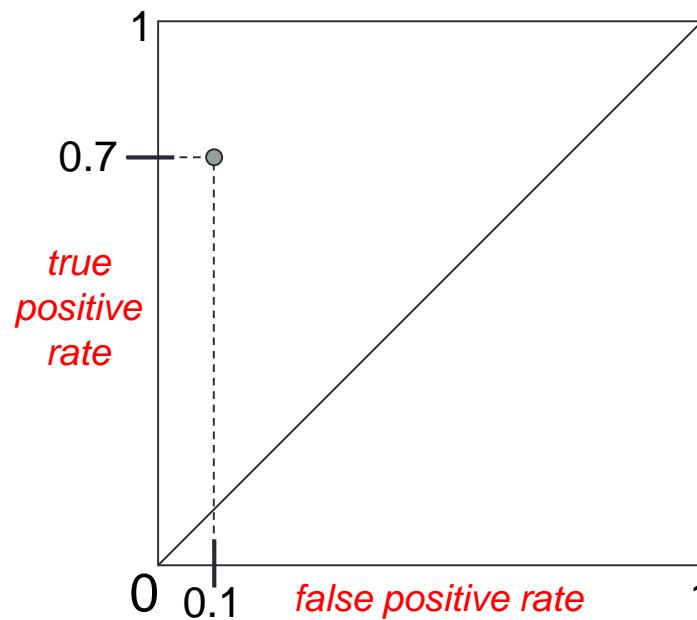
- Precision / Recall
- Sensitivity / Specificity
- Likelihood Ratios (LR+ and LR-)
- Positive / Negative Predictive Values
- F-Measure:
 - $F\alpha = [(1 + \alpha) (\text{Precision} \times \text{Recall})] / [(\alpha \times \text{Precision}) + \text{Recall}]$ $\alpha = 1, 2, 0.5$
- G-Mean: 2-class version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

Evaluating the output (Binary case)

How can we measure the performance of a deterministic classifier?

“recall”

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



*true
positive
rate*

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

1 - “specificity”

ROC Curve (Binary Case)

How can we measure the performance of a probability classifier?

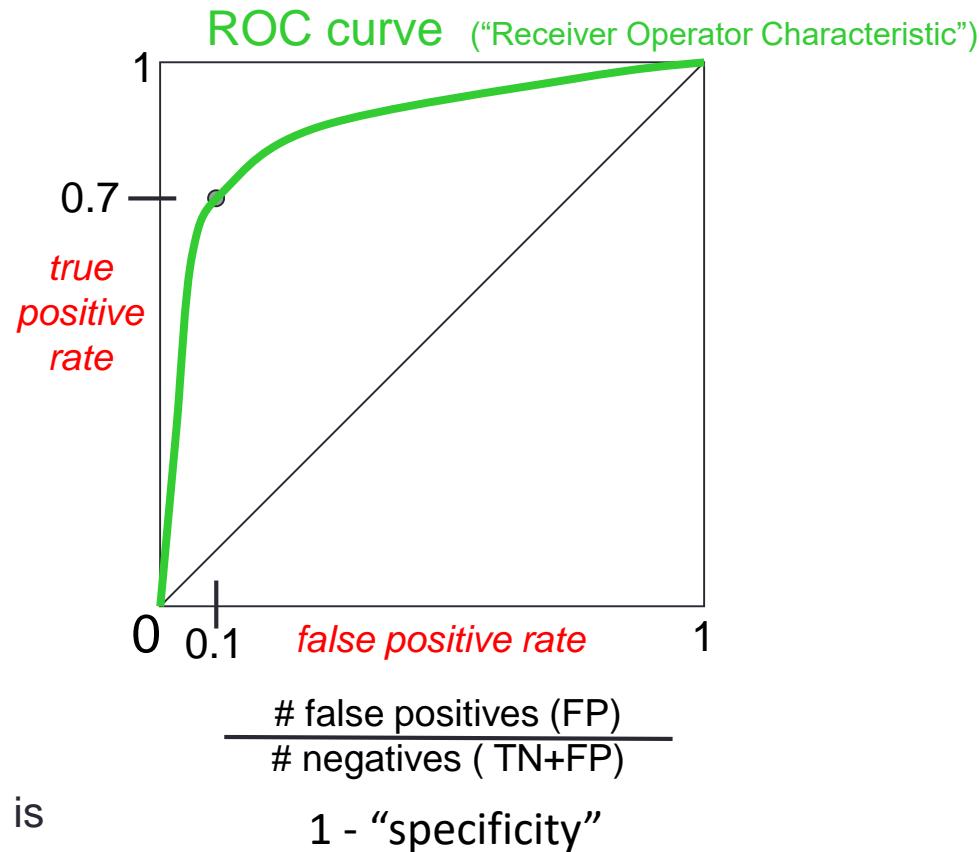
“recall”

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

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



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** = $TN/(FP+TN)$

UNTIL NOW

Linear Models

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

Classification: Perceptron

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

Error: $\|\text{sign}(\mathbf{w}^T \mathbf{x}) - y\|$

Algorithms: PLA / Pocket

Regression Lineal:

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

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

Algorithms: SVD/SGD

Logistic Regression:

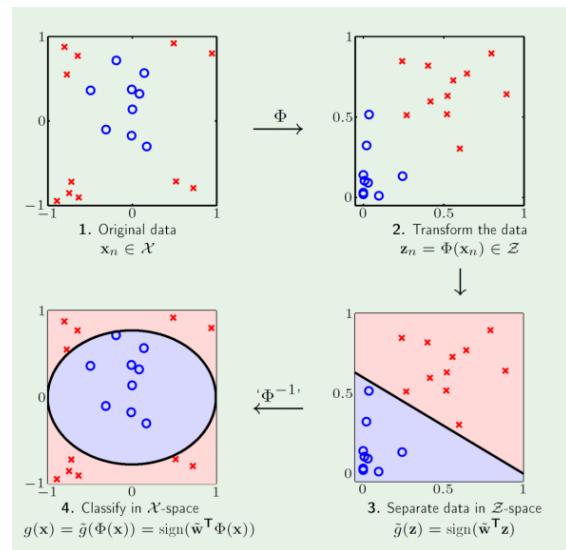
$$h(x) = \sigma(\mathbf{w}^T \mathbf{x})$$

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

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 $E_{out}(g) \approx 0$?