# Introducción al aprendizaje automático

•••

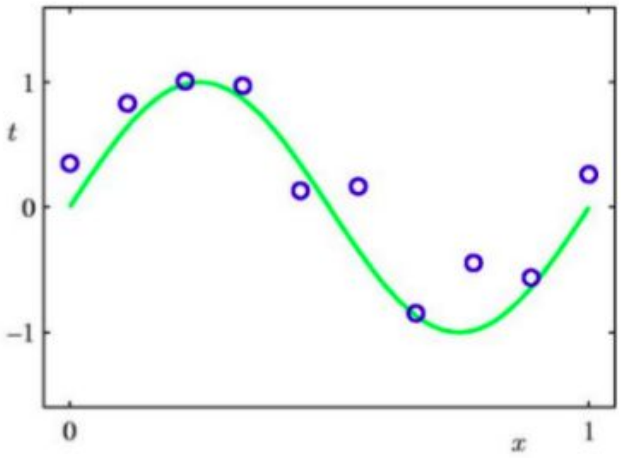
#2. Modelos probabilísticos y no paramétricos

# Regresión

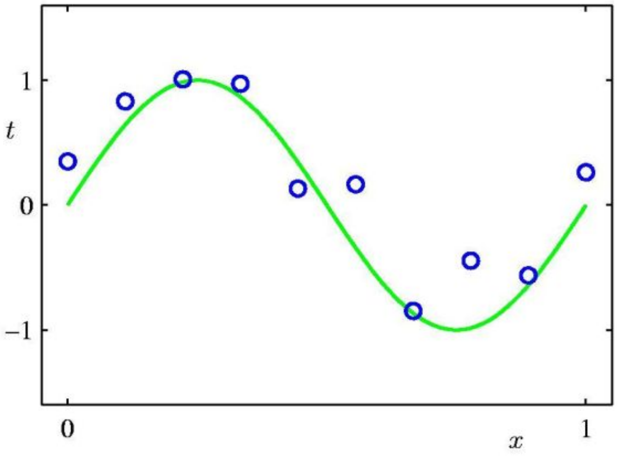
- Disponemos de $N$ pares de entrenamiento (observaciones)

$$\{(x_i, y_i)\}_{i=1}^{N} = \{(x_1, y_1), \cdots, (x_N, y_N)\}$$

- El problema de regresión consiste en estimar $f(x)$ a partir de estos datos



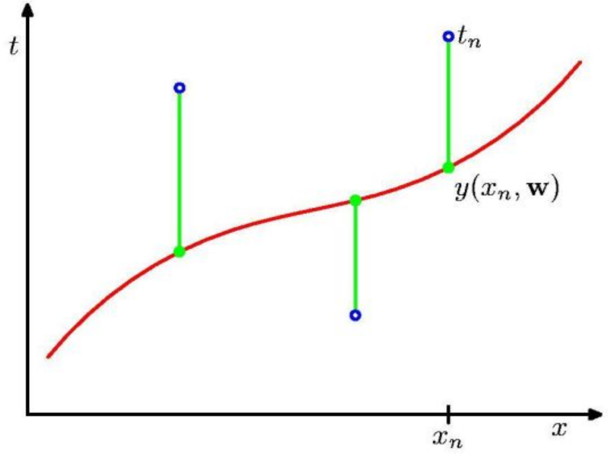
# Regresión polinomial

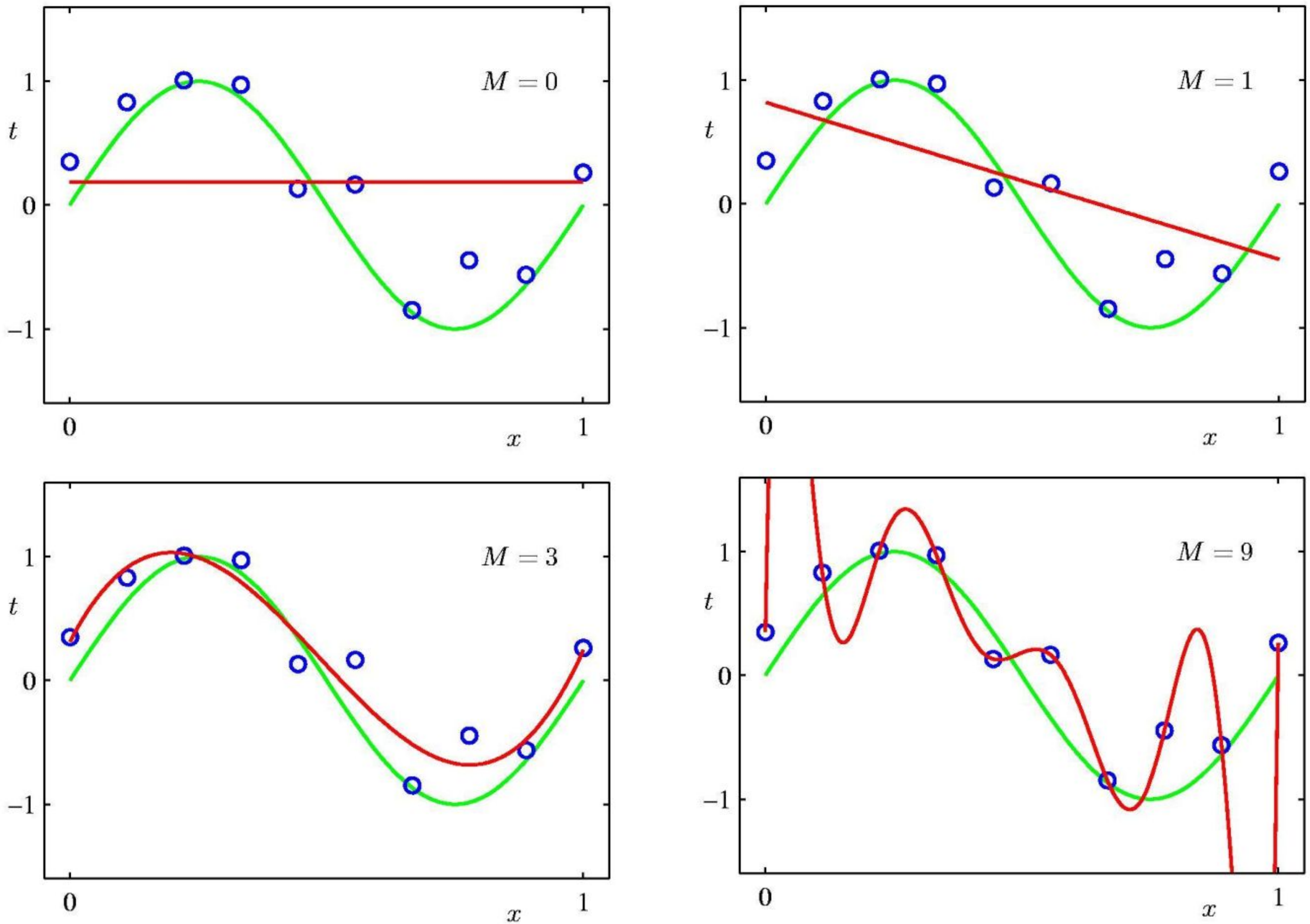



- En verde se ilustra la función "verdadera" (inaccesible)
- Las muestras son uniformes en $x$ y poseen ruido en $y$
- Utilizaremos una **<u>función de costo</u>** (error cuadrático) que mida el error en la predicción de $y$ mediante $f(x)$

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n,\mathbf{w}) - t_n\}^2$$

$$y(x,\mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$





M = 0
M = 1
M = 3
M = 9
t
x
1
0
−1

# Regresión lineal: solución de cuadrados mínimos

- Función de predicción lineal $y = f_w(x) = <x, w> = \sum_{k=1}^{K} x_k w_k$
- Función de costo: $L(w) = \sum_{i=1}^{N} (y^i - <x^i, w>)^2$
- Ecuaciones normales

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} x_1^1 & \dots & x_k^1 & \dots & x_K^1 \\ & & \vdots & & \\ x_1^N & \dots & x_k^N & \dots & x_K^N \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_k \\ \vdots \\ w_K \end{bmatrix}$$

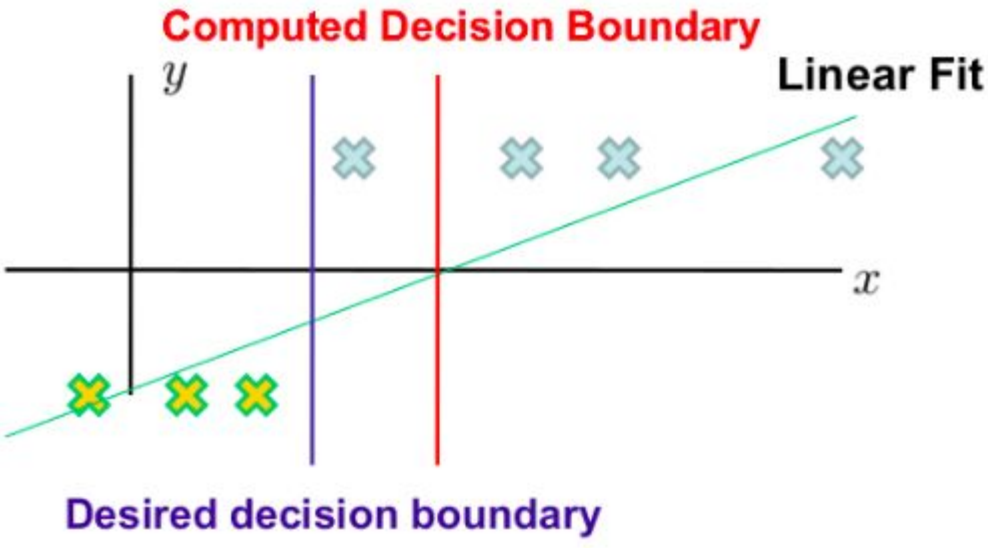
$$\mathbf{e} = \mathbf{y} - \mathbf{X}\mathbf{w}$$

$$L(\mathbf{w}) = \mathbf{e}^T\mathbf{e} \quad \rightarrow \quad \mathbf{w}^* = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

$$L(\mathbf{w}) = \mathbf{e}^T\mathbf{e} + \lambda\mathbf{w}^T\mathbf{w} \quad \rightarrow \quad \mathbf{w}^* = \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

# Error cuadrático en clasificación

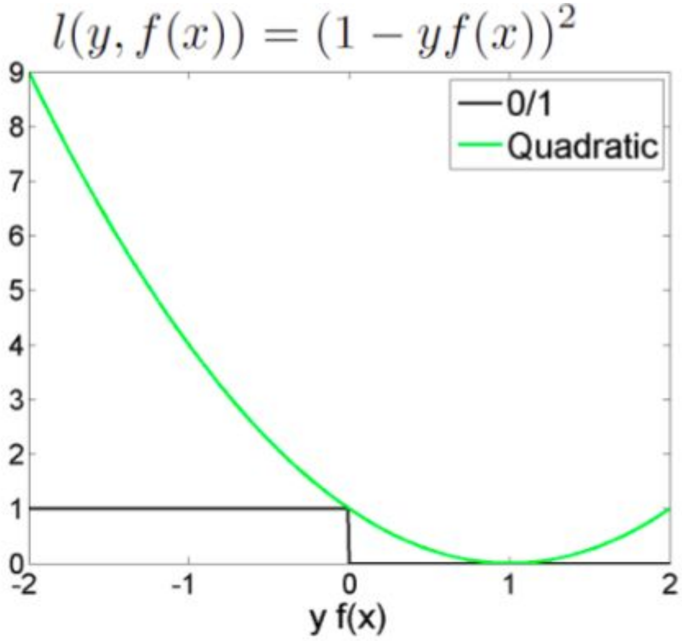
- Mínimo global único y solución en forma cerrada
- Pero, ¿es una medida del error de clasificación? ¿es adecuada?



# Error cuadrático en clasificación

$$y_{\pm} \in \{-1, 1\}$$

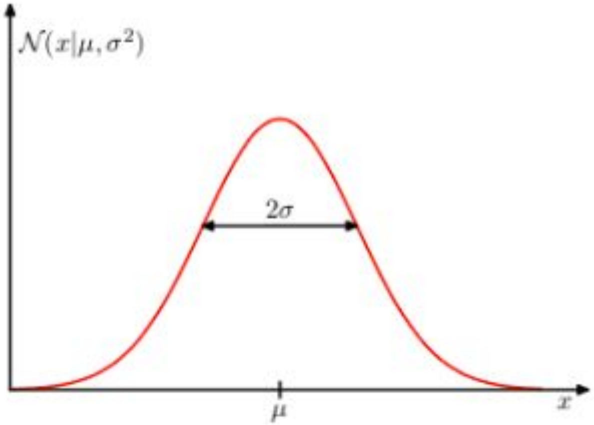
$$\begin{aligned} l(y, f(x)) &= (y - f(x))^2 \\ &\overset{y^2=1}{=} y^2(y - f(x))^2 \\ &= (y^2 - yf(x))^2 \\ &\overset{y^2=1}{=} (1 - yf(x))^2 \end{aligned}$$




- No es robusta frente a *outliers*
- Penaliza predicciones que son muy buenas

# Regresión polinomial. Solución por MV

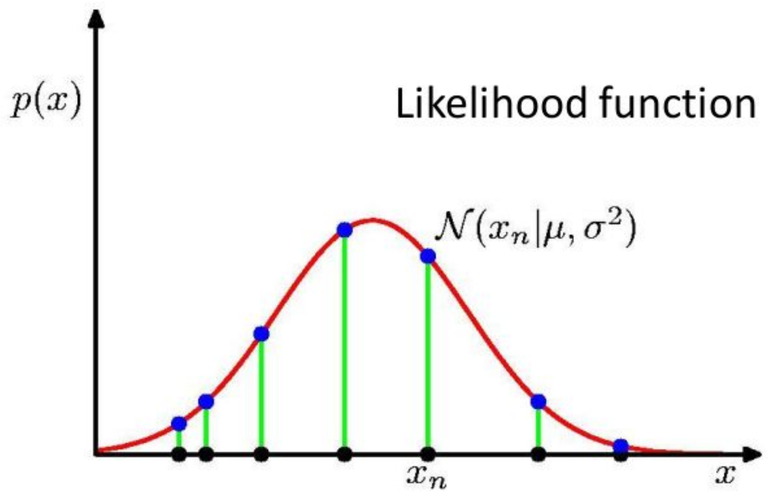
# Distribución gaussiana




$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} exp\left\{ -\frac{1}{2\sigma^2}(x-\mu)^2 \right\}$$

- Siempre positiva, integra a 1
- precisión $\beta = 1/\sigma^2$
- valor esperado $\mathbb{E}[x] = \int_{-\infty}^{\infty} \mathcal{N}(x|\mu,\sigma^2)x \, \mathrm{d}x = \mu$
- varianza $\mathrm{var}[x] = \mathbb{E}[x^2] - \mathbb{E}[x]^2 = \sigma^2$

# Máxima verosimilitud (MV)



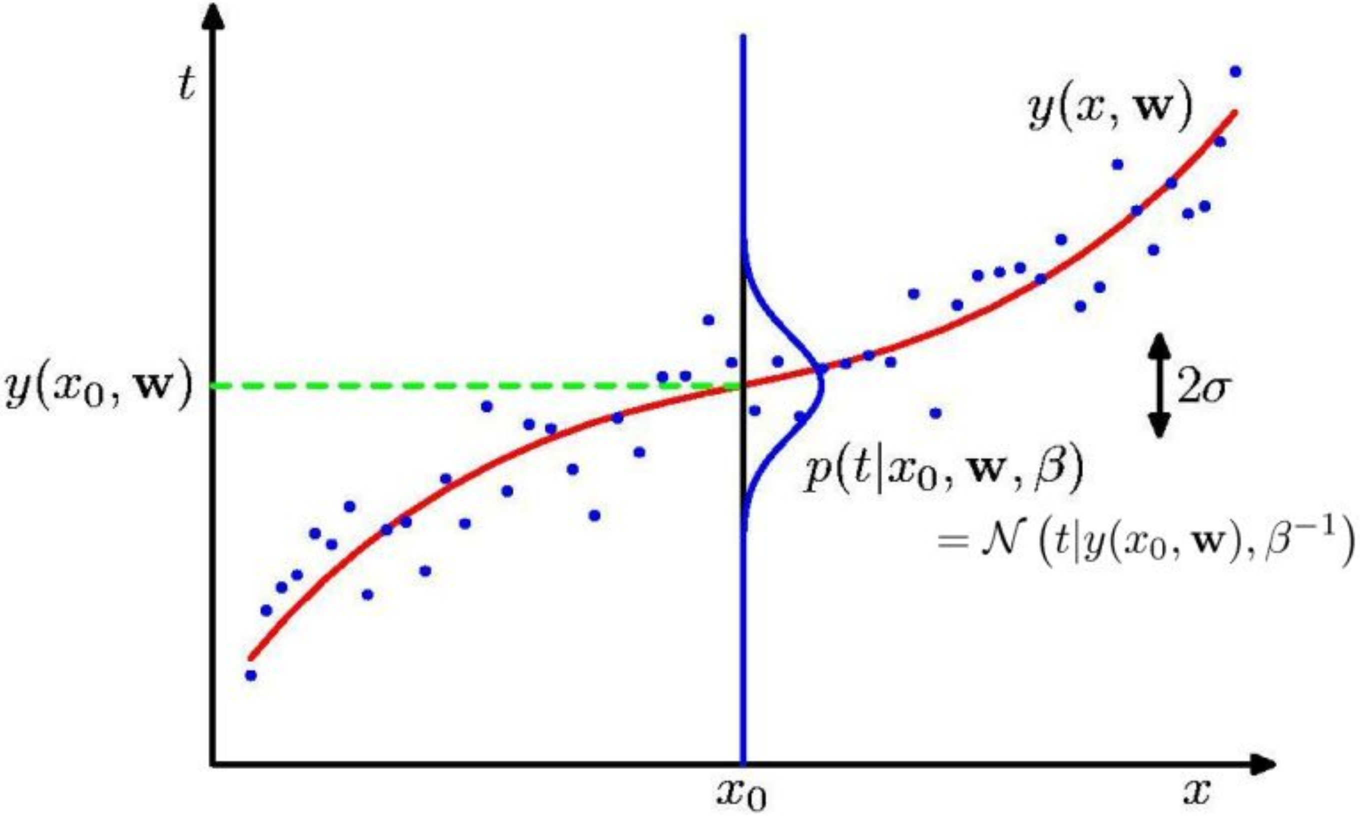

- Muestras iid
- Función de verosimilitud $p(\mathbf{x}|\mu, \sigma^2) = \prod_{n=1}^{N} \mathcal{N}(x_n|\mu, \sigma^2)$
- Logaritmo de la función de verosimilitud $\ln p(\mathbf{x}|\mu, \sigma^2) = -\frac{1}{2\sigma^2}\sum_{n=1}^{N}(x_n - \sigma)^2 - \frac{N}{2}\ln \sigma^2 - \frac{N}{2}\ln(2\pi)$
- Media muestral por MV $\mu_{ML} = \frac{1}{N}\sum_{n=1}^{N} x_n$
- Varianza muestral por MV $\sigma^2_{ML} = \frac{1}{N}\sum_{n=1}^{N}(x_n - \mu_{ML})^2$

# Revisando el ajuste de curvas

- Objetivo: predecir valores de salida $t$ para nuevas entradas $x$, en base a un conjunto de pares de entrenamiento $(x_1,t_1)$, … , $(x_N,t_N)$.

- Para capturar la incertidumbre sobre los valores de salida, podemos asumir que, dado un $x$, el valor de $t$ se genera a partir de una gaussiana de media $y(x; w)$ (la curva polinomial)

$$p(t|x, \mathbf{w}, \beta) = \mathcal{N}\left(t|y(x, \mathbf{w}), \beta^{-1}\right)$$



$t$
$y(x, \mathbf{w})$
$y(x_0, \mathbf{w})$
$2\sigma$
$p(t|x_0, \mathbf{w}, \beta)$
$= \mathcal{N}\left(t|y(x_0, \mathbf{w}), \beta^{-1}\right)$
$x_0$
$x$

# Probabilidades bayesianas

- Conocimiento "a priori" sobre los parámetros en $\mathrm{p}(w)$ (*prior*)
- Efecto de las observaciones $D=\{t_1, \dots t_N\}$ en el proceso de inferencia sobre $w$ se expresa mediante $\mathrm{p}(w|D)$ (*likelihood*)
- La incertidumbre sobre $w$ después de observar $D$ (*posterior*)

$$p(\mathbf{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{D})}$$

$$\text{posterior} \propto \text{likelihood} \times \text{prior}$$

- El denominador $\mathrm{p}(D)$ es un factor de normalización

# Revisando el ajuste de curvas

- Entrenamiento por MV, asumiendo muestras iid y distribución $p(t|x,w,\beta)$:

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}\left(t_n | y(x_n, \mathbf{w}, \beta^{-1}\right)$$

$$\ln p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta) = -\frac{\beta}{2} \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{N}{2} \ln \beta - \frac{N}{2} \ln (2\pi)$$

- La solución por MV, después de notar que los últimos dos términos no dependen de $w$ y que $\beta$ es un factor de escala, se obtiene de forma equivalente minimizando el error cuadrático medio:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} (y(n_n, \mathbf{w}) - t_n)^2$$

# Revisando el ajuste de curvas

- También podemos utilizar MV para estimar $\beta$:

$$\frac{1}{\beta_{ML}} = \frac{1}{N}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}_{ML}) - t_n\}^2$$

- Con w y $\beta$ podemos hacer predicciones sobre $x$ mediante la "distribución predictiva"

$$p(t|x, \mathbf{w}_{\mathrm{ML}}, \beta_{\mathrm{ML}}) = \mathcal{N}\left(t|y(x, \mathbf{w}_{\mathrm{ML}}), \beta_{\mathrm{ML}}^{-1}\right)$$

- Si consideramos un *prior* Gaussiano sobre $w$

$$p(\mathbf{w}|\alpha) = \mathcal{N}(\mathbf{w}|0, \alpha^{-1}\mathbf{I}) = \left(\frac{\alpha}{2\pi}\right)^{(M+1)/2}\exp\left\{-\frac{\alpha}{2}\mathbf{w}^T\mathbf{w}\right\}$$

# Máximo a posteriori (MAP)

- Posterior ∝ likelihood × prior

$$p(\mathbf{w}|\mathbf{x}, \mathbf{t}, \alpha, \beta) \propto p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \beta)p(\mathbf{w}|\alpha)$$

- Tomando el logaritmo de la función de verosimilitud de $p(w|x, t, \alpha, \beta)$ y considerando como antes sólo los términos que dependen de $w$

$$\frac{\beta}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{\alpha}{2}\mathbf{w}^T\mathbf{w}$$

resulta en error cuadrático con regularización $L_2$ de parámetro $\lambda=\alpha/\beta$

# Regresión logística

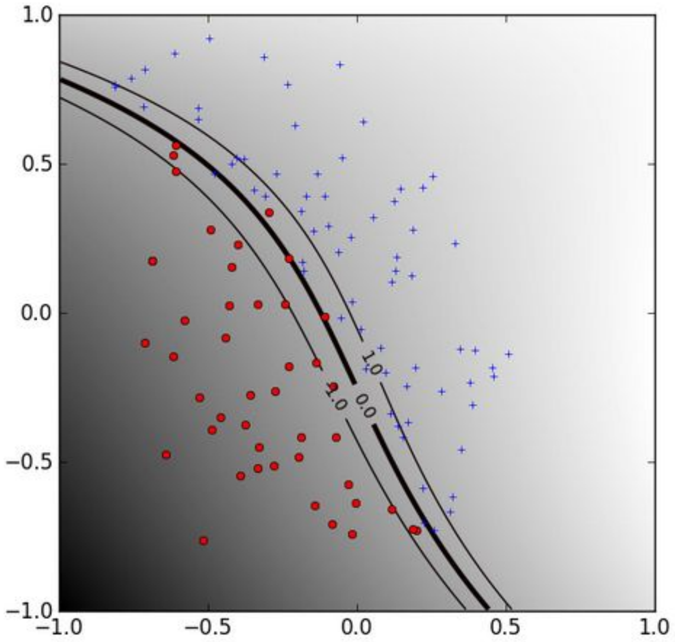
# Clasificación basada en probabilidades

- Objetivo: dar la probabilidad de que una instancia $x$ sea de una clase $y$, es decir, aprender $\mathrm{p}(y|x)$

- Recordar:

$$0 \leq \mathrm{p}(evento) \leq 1$$

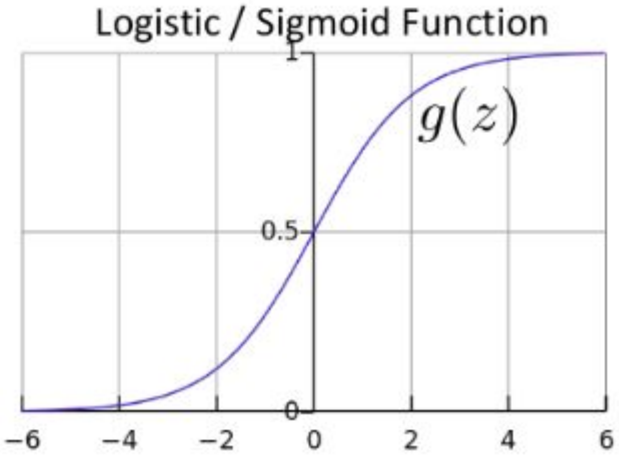
$$\mathrm{p}(evento) + \mathrm{p}(\neg evento) = 1$$



# Regresión logística

- Aproximación probabilística al problema de clasificación
- La función de predicción $h_w(x)$ debe dar una aproximación de $p(y=1|x,w)$
- $0 \leq h_w(x) \leq 1$

$$h_w(x) = g(w^T x) = \frac{1}{1 + \exp(-w^T x)}$$



# Regresión logística

- Given $\left\{\left(\boldsymbol{x}^{(1)}, y^{(1)}\right),\left(\boldsymbol{x}^{(2)}, y^{(2)}\right), \ldots,\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}$
  where $\boldsymbol{x}^{(i)} \in \mathbb{R}^d,\ y^{(i)} \in \{0,1\}$

- Model: $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = g\left(\boldsymbol{\theta}^\top \boldsymbol{x}\right)$

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

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \qquad \boldsymbol{x}^\top = \begin{bmatrix} 1 & x_1 & \ldots & x_d \end{bmatrix}$$

# Regresión logística. Función de costo

- Can't just use squared loss as in linear regression:

$$J(\boldsymbol{\theta}) = \frac{1}{2n} \sum_{i=1}^{n} \left( h_{\boldsymbol{\theta}} \left( \boldsymbol{x}^{(i)} \right) - y^{(i)} \right)^2$$

  – Using the logistic regression model

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}} \boldsymbol{x}}}$$

  results in a non-convex optimization

# Regresión logística. Función de costo

Training set: $\{(\mathbf{x}^1, y^1), \ldots, (\mathbf{x}^N, y^N)\}, \quad \mathbf{x} \in R^M, \quad y \in \{0,1\}$

y: discrete observations: model as samples from Bernoulli distribution

$$\begin{aligned} P(y=1|\mathbf{x},\mathbf{w}) &= f(\mathbf{x},\mathbf{w}) \\ P(y=0|\mathbf{x},\mathbf{w}) &= 1 - f(\mathbf{x},\mathbf{w}) \end{aligned}$$

$$P(y|\mathbf{x}) = (f(\mathbf{x},\mathbf{w}))^y \, (1 - f(\mathbf{x},\mathbf{w}))^{1-y}$$

Find w that maximizes the likelihood of labels in the training set

$$\begin{aligned} -L(\mathbf{w}) = C(\mathbf{w}) = & \quad \log P(\mathbf{y}|\mathbf{X},\mathbf{w}) = \sum_{i=1}^{N} \log P(y^i|\mathbf{x}^i,\mathbf{w}) \\ = & \quad \sum_i y^i \log f(\mathbf{x}^i,\mathbf{w}) + (1-y^i)\log(1 - f(\mathbf{x}^i,\mathbf{w})) \end{aligned}$$

# Intuition Behind the Objective

$$J(\boldsymbol{\theta}) = -\sum_{i=1}^{n} \left[ y^{(i)} \log h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) + \left(1 - y^{(i)}\right) \log \left(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})\right) \right]$$

- Cost of a single instance:

$$\text{cost}\left(h_{\boldsymbol{\theta}}(\boldsymbol{x}), y\right) = \begin{cases} -\log(h_{\boldsymbol{\theta}}(\boldsymbol{x})) & \text{if } y = 1 \\ -\log(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x})) & \text{if } y = 0 \end{cases}$$

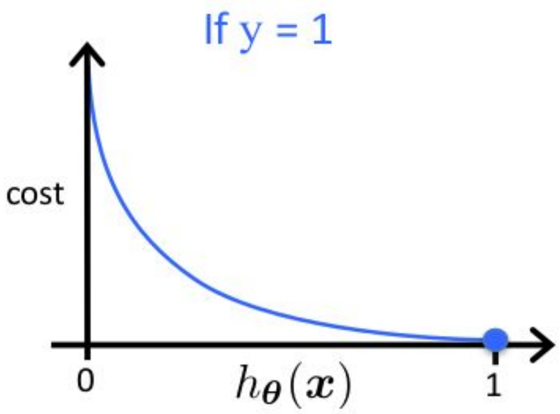
- Can re-write objective function as

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{n} \text{cost}\left(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}), y^{(i)}\right)$$

Compare to linear regression: $J(\boldsymbol{\theta}) = \frac{1}{2n} \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2$

# Intuition Behind the Objective

$$\text{cost}\,(h_{\boldsymbol{\theta}}(\boldsymbol{x}), y) = \begin{cases} \boxed{-\log(h_{\boldsymbol{\theta}}(\boldsymbol{x})) \quad \text{if } y = 1} \\ -\log(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x})) \quad \text{if } y = 0 \end{cases}$$

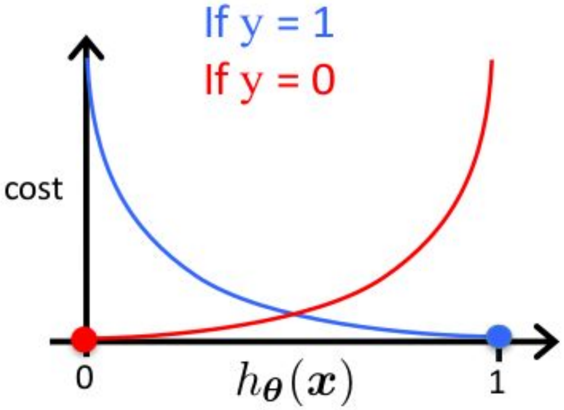



If y = 1

- Cost = 0 if prediction is correct
- As $h_{\boldsymbol{\theta}}(\boldsymbol{x}) \to 0, \text{cost} \to \infty$
- Captures intuition that larger mistakes should get larger penalties
  - e.g., predict $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = 0$, but y = 1

# Intuition Behind the Objective

$$\text{cost}\left(h_{\boldsymbol{\theta}}(\boldsymbol{x}), y\right) = \begin{cases} \boxed{-\log(h_{\boldsymbol{\theta}}(\boldsymbol{x})) \quad \text{if } y = 1} \\ \boxed{-\log(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x})) \quad \text{if } y = 0} \end{cases}$$




If y = 0

- Cost = 0 if prediction is correct
- As $(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x})) \rightarrow 0, \text{cost} \rightarrow \infty$
- Captures intuition that larger mistakes should get larger penalties

# Regularized Logistic Regression

$$J(\boldsymbol{\theta}) = -\sum_{i=1}^{n}\left[y^{(i)}\log h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) + \left(1 - y^{(i)}\right)\log\left(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})\right)\right]$$

- We can regularize logistic regression exactly as before:

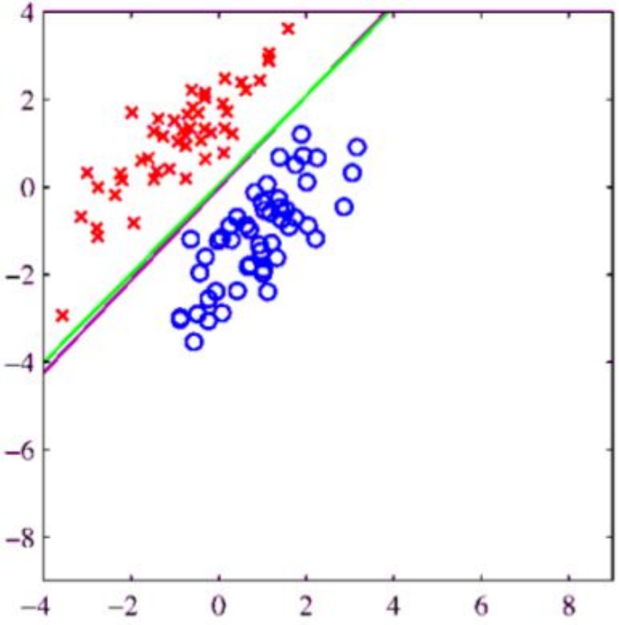
$$\begin{aligned} J_{\text{regularized}}(\boldsymbol{\theta}) &= J(\boldsymbol{\theta}) + \lambda\sum_{j=1}^{d}\theta_j^2 \\ &= J(\boldsymbol{\theta}) + \lambda\|\boldsymbol{\theta}_{[1:d]}\|_2^2 \end{aligned}$$

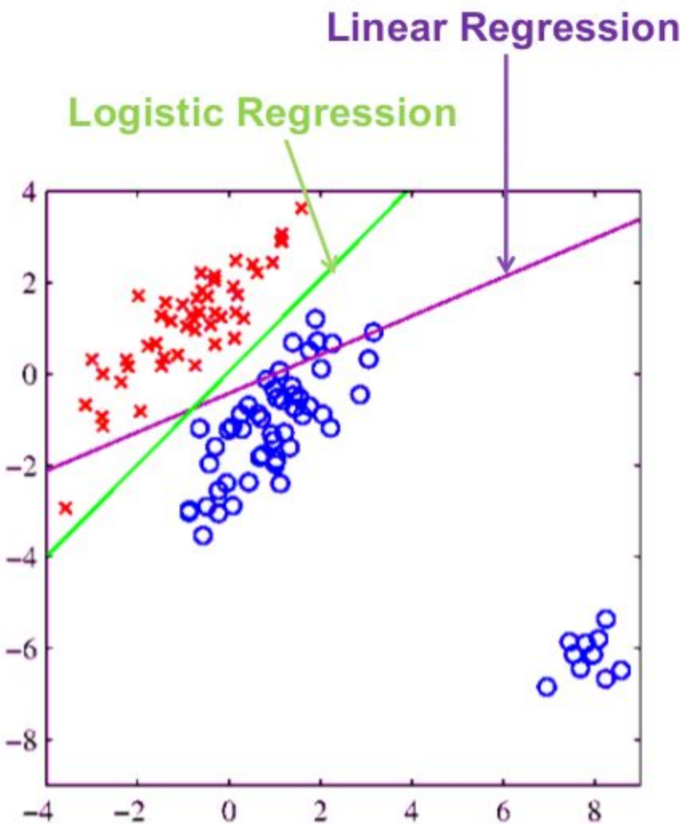
[1:d] => exclude the bias!

$$\theta^* = \arg\min_{\theta} J(\theta)$$

# Logistic vs Linear Regression

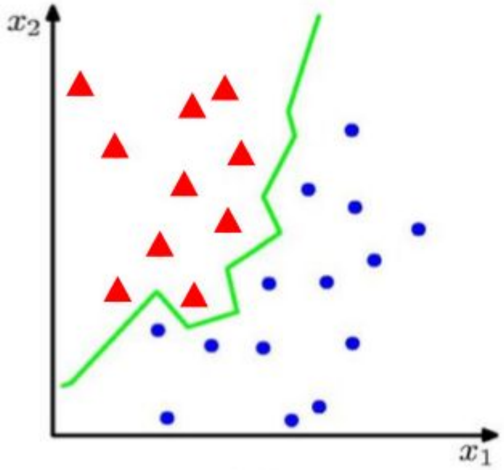
Logistic regression is more robust



# Modelos no paramétricos: vecinos más cercanos

# Classification




- Suppose we are given a training set of N observations

$$(x_1, \ldots, x_N) \text{ and } (y_1, \ldots, y_N), x_i \in \mathbb{R}^d, y_i \in \{-1, 1\}$$

- Classification problem is to estimate f(x) from this data such that

$$f(x_i) = y_i$$

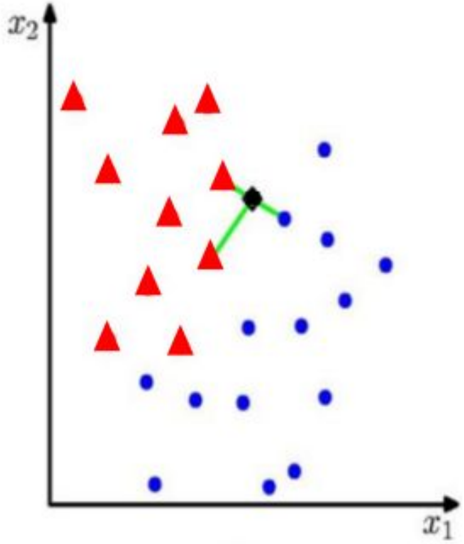
# K Nearest Neighbour (K-NN) Classifier

## Algorithm

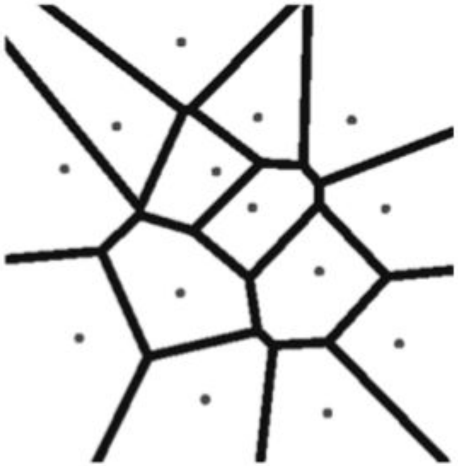
- For each test point, x, to be classified, find the K nearest samples in the training data
- Classify the point, x, according to the majority vote of their class labels

e.g. K = 3

• applicable to multi-class case

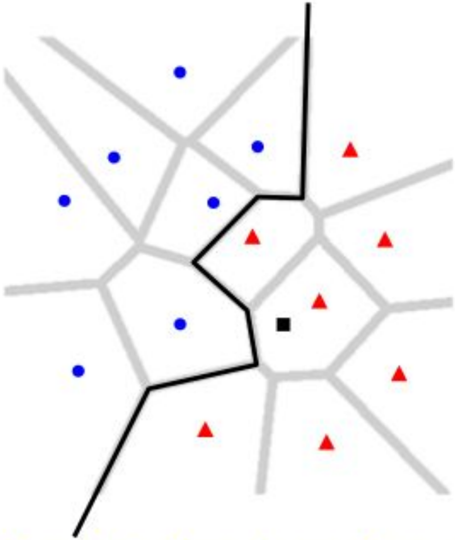


# K = 1



Voronoi diagram:

- partitions the space into regions
- boundaries are equal distance from training points



Classification boundary:

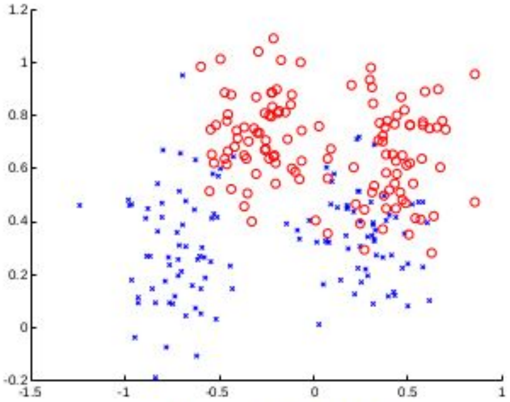
- non-linear

# A sampling assumption: training and test data

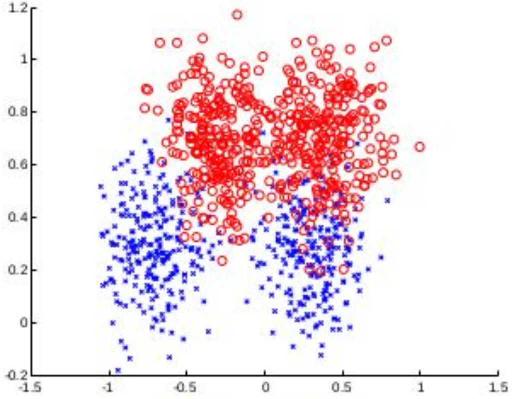
• Assume that the training examples are drawn independently from the set of all possible examples.

• This makes it very unlikely that a strong regularity in the training data will be absent in the test data.

• Measure classification error as $= \frac{1}{N}\sum_{i=1}^{N} \underbrace{[\mathbf{y}_i \neq f(\mathbf{x}_i)]}_{\text{loss function}}$ The "risk"



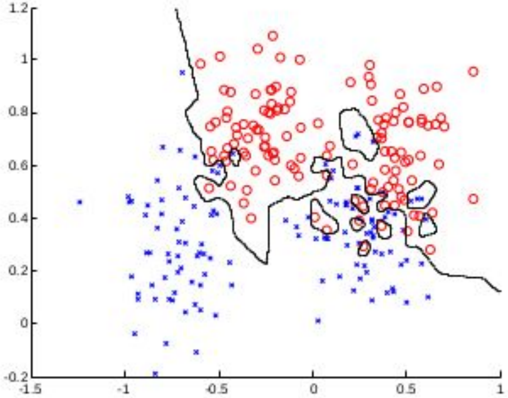
Training data



Testing data

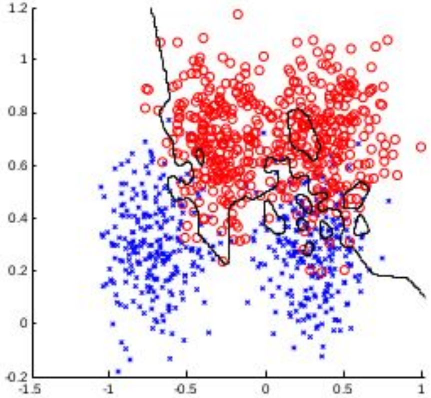
# K = 1

Training data




error = 0.0

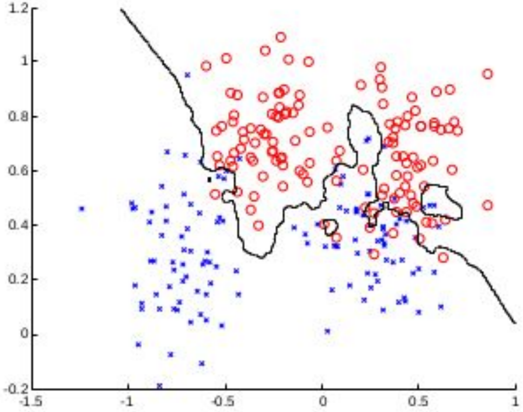
Testing data




error = 0.15

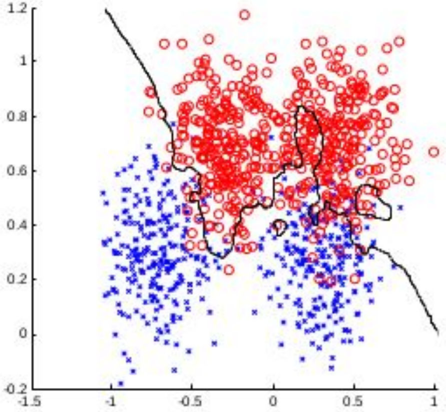
# K = 3

Training data



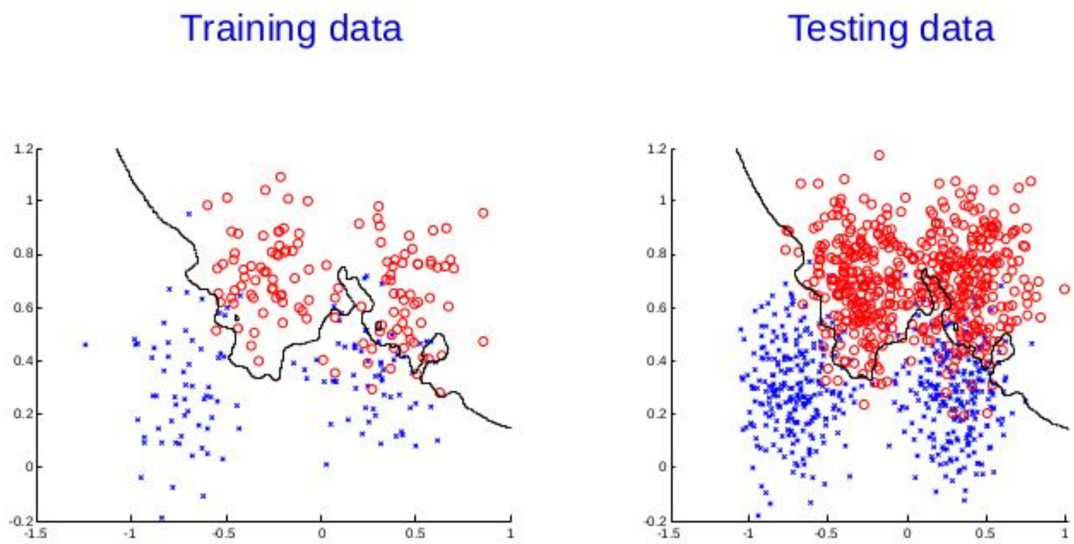

error = 0.0760

Testing data




error = 0.1340

# K = 7

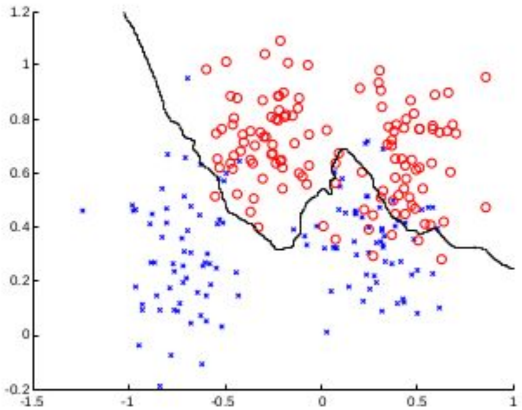



error = 0.1320

error = 0.1110

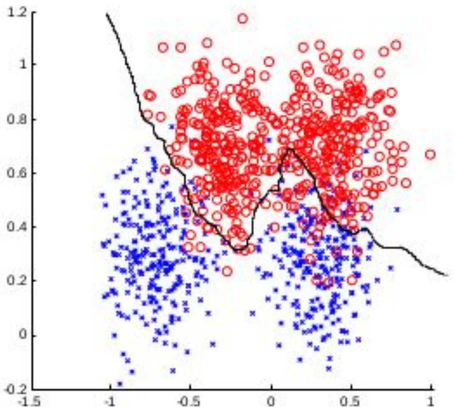
# K = 21

Training data




error = 0.1120

Testing data




error = 0.0920

# Properties and training

As K increases:

- Classification boundary becomes smoother
- Training error can increase

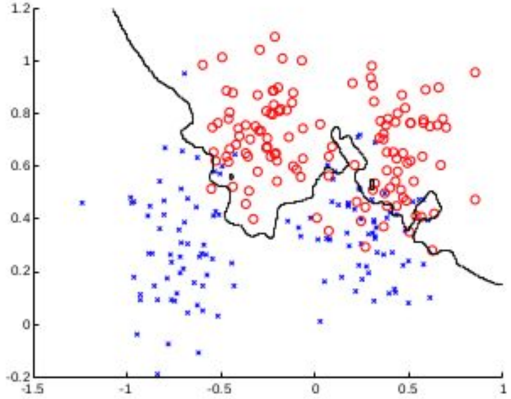
Choose (learn) K by cross-validation

- Split training data into training and validation
- Hold out validation data and measure error on this

# Summary

## Advantages:

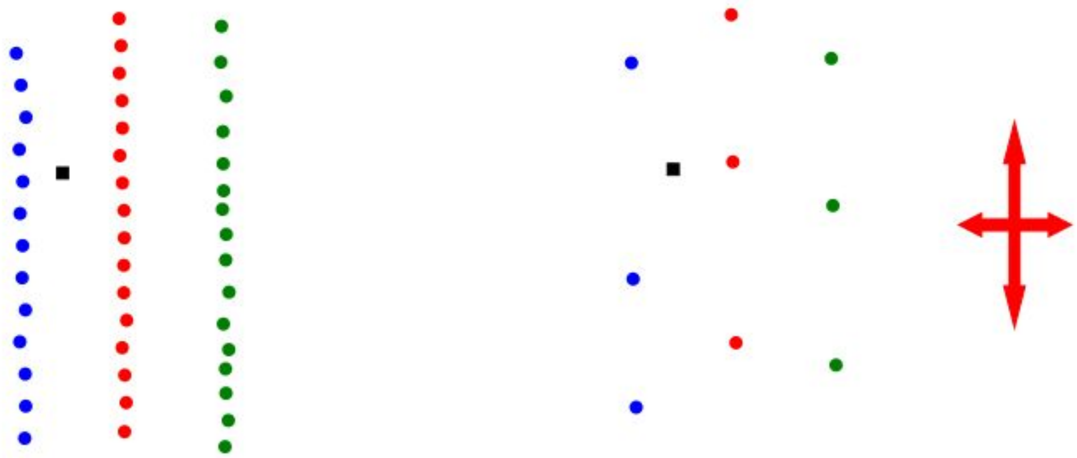
- K-NN is a simple but effective classification procedure
- Applies to multi-class classification
- Decision surfaces are non-linear
- Quality of predictions automatically improves with more training data
- Only a single parameter, K; easily tuned by cross-validation



# Summary

## Disadvantages:

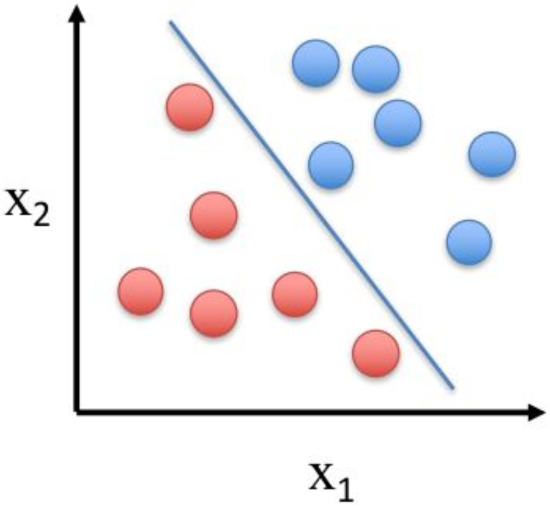
- What does nearest mean? Need to specify a distance metric.
- Computational cost: must store and search through the entire training set at test time. Can alleviate this problem by thinning, and use of efficient data structures like KD trees.



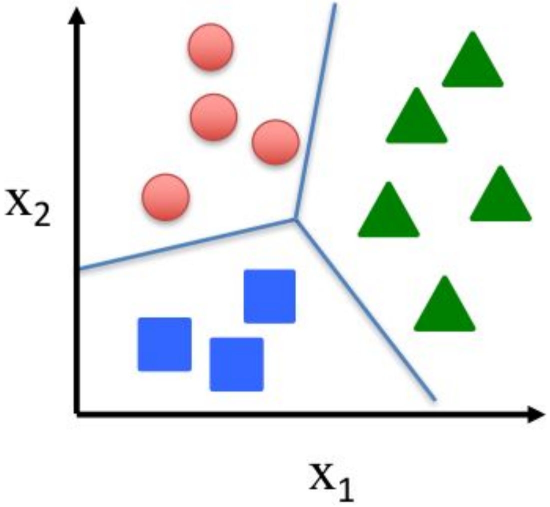
# Problemas multiclase

# Multi-Class Classification

Binary classification:




Multi-class classification:




Disease diagnosis: healthy / cold / flu / pneumonia

Object classification: desk / chair / monitor / bookcase

# Multi-Class Logistic Regression

- For 2 classes:

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^\mathsf{T}\boldsymbol{x})} = \frac{\exp(\boldsymbol{\theta}^\mathsf{T}\boldsymbol{x})}{\boxed{1} + \boxed{\exp(\boldsymbol{\theta}^\mathsf{T}\boldsymbol{x})}}$$

weight assigned to y = 0

weight assigned to y = 1

- For C classes {1, ..., C}:

$$p(y = c \mid \boldsymbol{x}; \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_C) = \frac{\exp(\boldsymbol{\theta}_c^\mathsf{T}\boldsymbol{x})}{\sum_{c=1}^{C} \exp(\boldsymbol{\theta}_c^\mathsf{T}\boldsymbol{x})}$$

  – Called the **softmax** function

# Implementing Multi-Class Logistic Regression

- Use $h_c(\boldsymbol{x}) = \frac{\exp(\boldsymbol{\theta}_c^\mathsf{T}\boldsymbol{x})}{\sum_{c=1}^{C}\exp(\boldsymbol{\theta}_c^\mathsf{T}\boldsymbol{x})}$ as the model for class c

- Gradient descent simultaneously updates all parameters for all models
  - Same derivative as before, just with the above $h_c(\mathbf{x})$

- Predict class label as the most probable label

$$\max_c h_c(\boldsymbol{x})$$

# What is multiclass classification?

- An input can belong to one of K classes

- Training data: Input associated with class label (a number from 1 to K)
- Prediction: Given a new input, predict the class label

Each input belongs to exactly one class. Not more, not less.

- Otherwise, the problem is not multiclass classification
- If an input can be assigned multiple labels (think tags for emails rather than folders), it is called *multi-label classification*

# Binary to multiclass

- Can we use a binary classifier to construct a multiclass classifier?
  - Decompose the prediction into multiple binary decisions
- How to decompose?
  - One-vs-all
  - All-vs-all
  - Error correcting codes

# 1. One-vs-all classification

- Assumption: Each class individually separable from ***all*** the others

- Learning: Given a dataset $D = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$ $\quad$ $\boldsymbol{x} \in \Re^n$, $\boldsymbol{y} \in \{1,2,\cdots,K\}$
  - Decompose into K binary classification tasks
  - For class k, construct a binary classification task as:
    - **Positive examples**: Elements of D with label k
    - **Negative examples**: All other elements of D
  - Train K binary classifiers $\mathbf{w}_1, \mathbf{w}_2, \cdots \mathbf{w}_K$ using any learning algorithm we have seen

# 1. One-vs-all classification

- Assumption: Each class individually separable from ***all*** the others

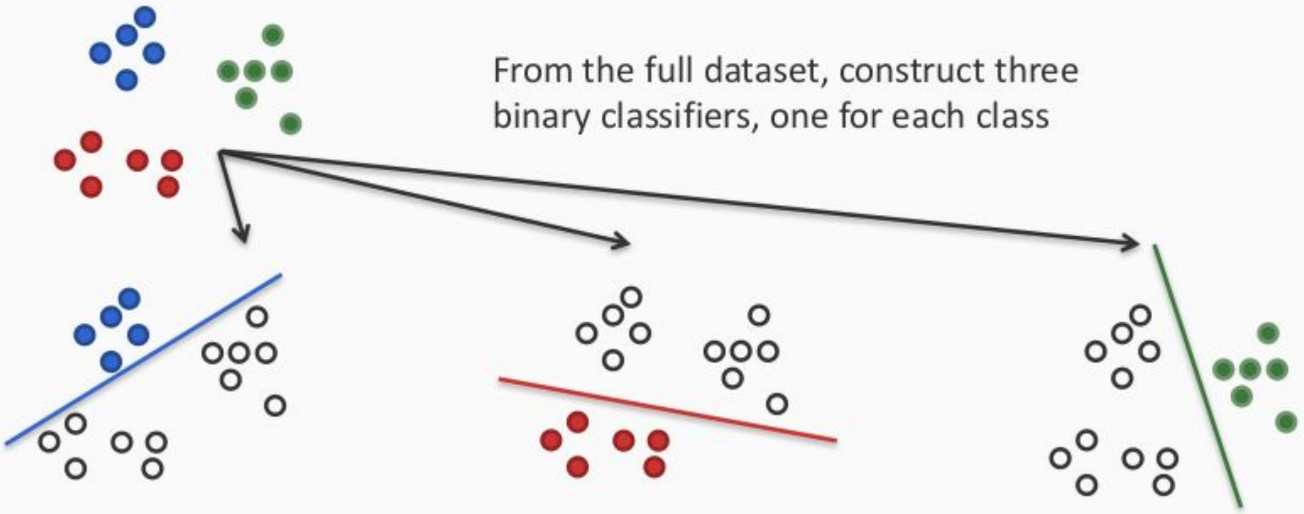
- Learning: Given a dataset $D = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$ $\quad$ $\boldsymbol{x} \in \Re^n$, $\boldsymbol{y} \in \{1,2,\cdots,K\}$
  - Train K binary classifiers $\mathbf{w}_1, \mathbf{w}_2, \cdots \mathbf{w}_K$ using any learning algorithm we have seen

- Prediction: "*Winner Takes All*"

$$\text{argmax}_i \; \mathbf{w}_i^T \mathbf{x}$$

# Visualizing One-vs-all

From the full dataset, construct three binary classifiers, one for each class

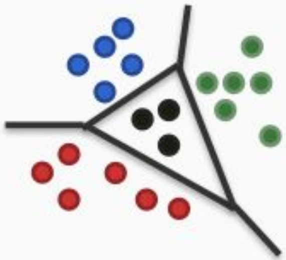


$\mathbf{w}_{blue}^T\mathbf{x} > 0$
for **blue** **inputs**

$\mathbf{w}_{red}^T\mathbf{x} > 0$
for **red** **inputs**

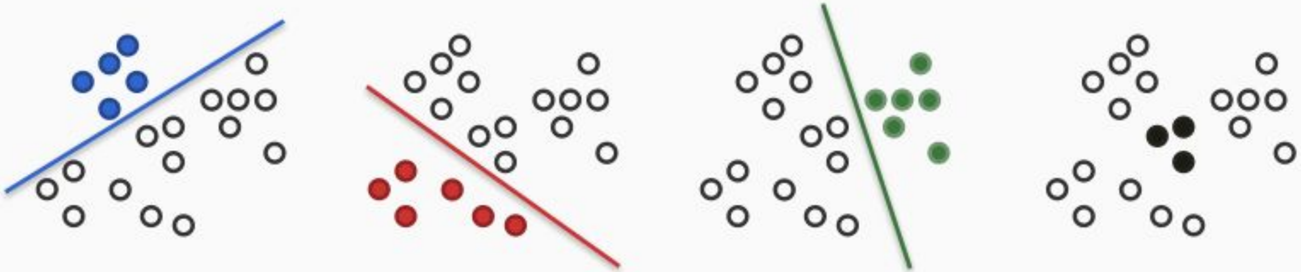
$\mathbf{w}_{green}^T\mathbf{x} > 0$
for **green** **inputs**

# One-vs-all may not always work



Black points are not separable with a single binary classifier

*The decomposition will not work for these cases!*



$\mathbf{w}_{\text{blue}}^T\mathbf{x} > 0$ for **blue inputs**

$\mathbf{w}_{\text{red}}^T\mathbf{x} > 0$ for **red inputs**

$\mathbf{w}_{\text{green}}^T\mathbf{x} > 0$ for **green inputs**

**???**

# 2. All-vs-all classification

Sometimes called one-vs-one

- Assumption: *Every* pair of classes is separable

- Learning: Given a dataset $D = \{(\boldsymbol{x_i}, \boldsymbol{y}_i)\}$, $\quad \boldsymbol{x} \in \Re^n$, $\boldsymbol{y} \in \{1,2,\cdots,K\}$
  - For every pair of labels (j, k), create a binary classifier with:
    - Positive examples: All examples with label j
    - Negative examples: All examples with label k
  - Train $\binom{K}{2} = \frac{K(K-1)}{2}$ classifiers to separate every pair of labels from each other

# 2. All-vs-all classification

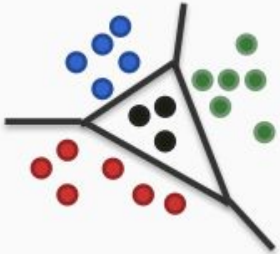
Sometimes called one-vs-one

- Assumption: *Every* pair of classes is separable

- Learning: Given a dataset $D = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$, $\quad \boldsymbol{x} \in \Re^n$, $\boldsymbol{y} \in \{1,2,\cdots,K\}$
  - Train $\binom{K}{2} = \frac{K(K-1)}{2}$ classifiers to separate every pair of labels from each other

- Prediction: More complex, each label get K-1 votes
  - How to combine the votes? Many methods
    - Majority: Pick the label with maximum votes
    - Organize a tournament between the labels

# All-vs-all classification



- Every pair of labels is linearly separable here
  – When a pair of labels is considered, all others are ignored
- Problems
  1. $O(K^2)$ weight vectors to train and store
  2. Size of training set for a pair of labels could be very small, leading to overfitting of the binary classifiers
  3. Prediction is often ad-hoc and might be unstable

     Eg: What if two classes get the same number of votes? For a tournament, what is the sequence in which the labels compete?

# 3. Error correcting output codes (ECOC)

- Each binary classifier provides one bit of information
- With K labels, we only need $\log_2 K$ bits
  - One-vs-all uses K bits (one per classifier)
  - All-vs-all uses $O(K^2)$ bits
- Can we get by with O(log K) classifiers?
  - Yes! Encode each label as a binary string
  - Or alternatively, if we do train more than O(log K) classifiers, can we use the redundancy to improve classification accuracy?

# Using $\log_2 K$ classifiers

| label# | Code | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 2 | 0 | 1 | 0 |
| 3 | 0 | 1 | 1 |
| 4 | 1 | 0 | 0 |
| 5 | 1 | 0 | 1 |
| 6 | 1 | 1 | 0 |
| 7 | 1 | 1 | 1 |

8 classes, code-length = 3

- Learning:
  - Represent each label by a bit string
  - Train one binary classifier for each bit
- Prediction:
  - Use the predictions from all the classifiers to create a $\log_2 N$ bit string that uniquely decides the output
- What could go wrong here?
  - Even if one of the classifiers makes a mistake, final prediction is wrong!

# Error correcting output coding

| # | Code | | | | |
|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 0 | 0 |
| **1** | 0 | 0 | 1 | 1 | 0 |
| **2** | 0 | 1 | 0 | 1 | 1 |
| **3** | 0 | 1 | 1 | 0 | 1 |
| **4** | 1 | 0 | 0 | 1 | 1 |
| **5** | 1 | 0 | 1 | 0 | 0 |
| **6** | 1 | 1 | 0 | 0 | 0 |
| **7** | 1 | 1 | 1 | 1 | 1 |

8 classes, code-length = 5

*Answer: Use redundancy*

- Assign a binary string with each label
  - Could be random
  - Length of the code word **L** >= $\log_2 K$ is a parameter

- Train one binary classifier for each bit
  - Effectively, split the data into random dichotomies
  - We need only $\log_2 K$ bits
    - Additional bits act as an error correcting code

- One-vs-all is a special case.
  - How?

# How to predict?

| # | Code | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 2 | 0 | 1 | 0 | 1 | 1 |
| 3 | 0 | 1 | 1 | 0 | 1 |
| 4 | 1 | 0 | 0 | 1 | 1 |
| 5 | 1 | 0 | 1 | 0 | 0 |
| 6 | 1 | 1 | 0 | 0 | 0 |
| 7 | 1 | 1 | 1 | 1 | 1 |

8 classes, code-length = 5

- Prediction
  - Run all **L** binary classifiers on the example
  - Gives us a predicted bit string of length **L**
  - Output = label whose code word is "closest" to the prediction
  - Closest defined using Hamming distance
    - Longer code length is better, better error-correction

- Example
  - Suppose the binary classifiers here predict 11010
  - The closest label to this is 6, with code word 11000

# Error correcting codes: Discussion

- Assumes that columns are independent
  - Otherwise, ineffective encoding
- Strong theoretical results that depend on code length
  - If minimal Hamming distance between two rows is d, then the prediction can correct up to (d-1)/2 errors in the binary predictions
- Code assignment could be random, or designed for the dataset/task
- One-vs-all and all-vs-all are special cases
  - All-vs-all needs a ternary code (not binary)