

Introducción al aprendizaje automático

...

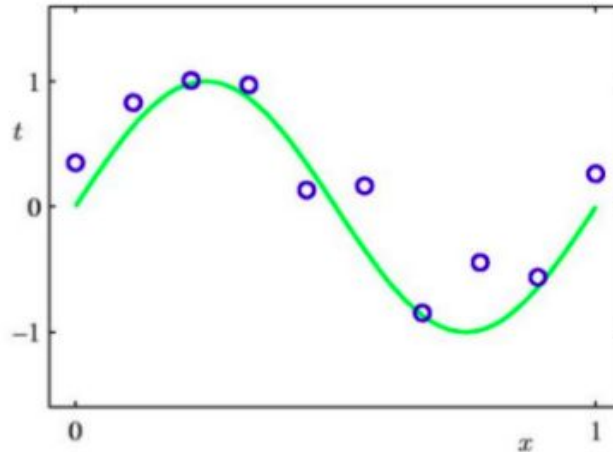
#2. Modelos probabilísticos y no paramétricos

Regresión

- Disponemos de N pares de entrenamiento (observaciones)

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

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

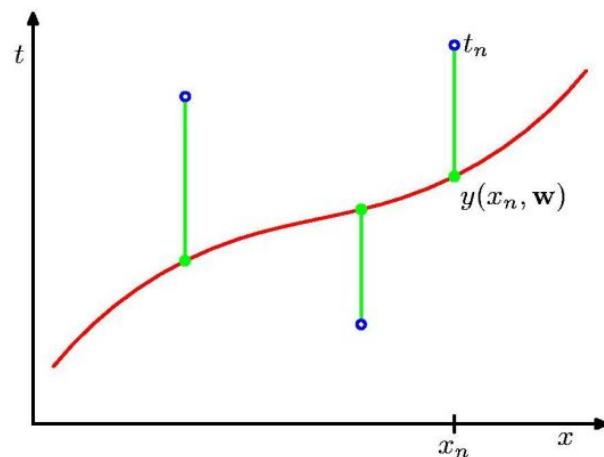
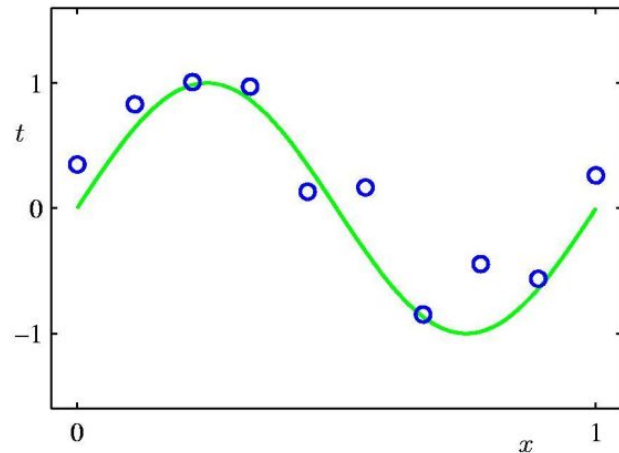


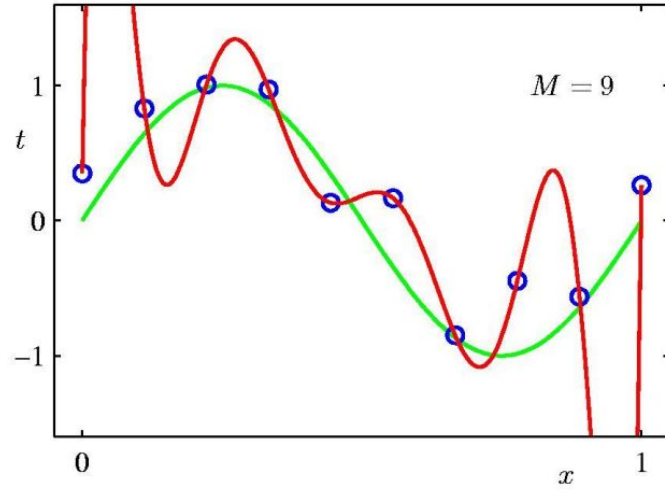
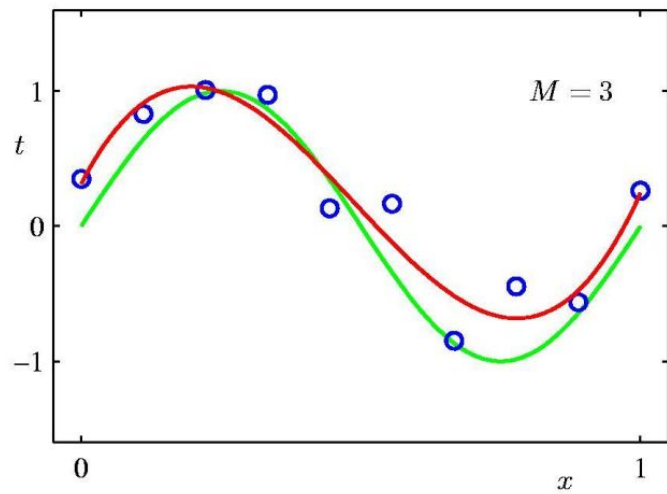
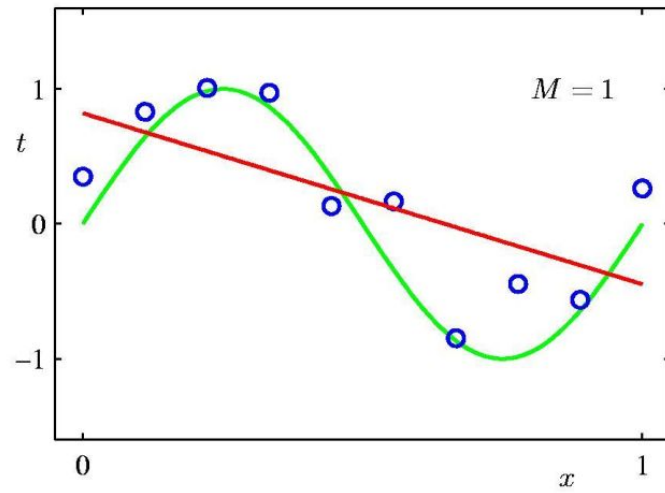
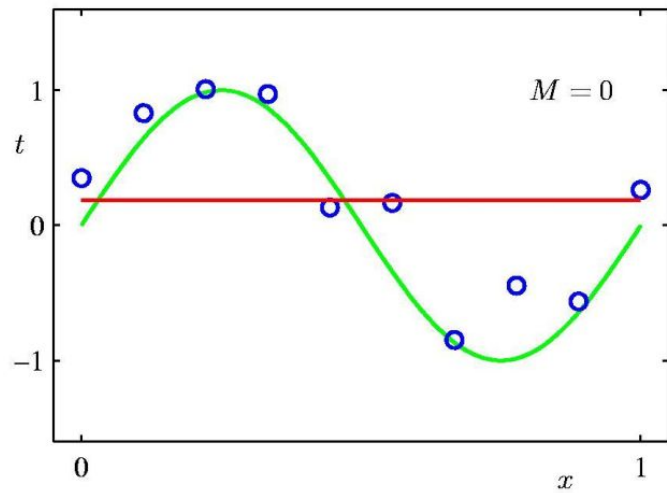
Regresión polinomial

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

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

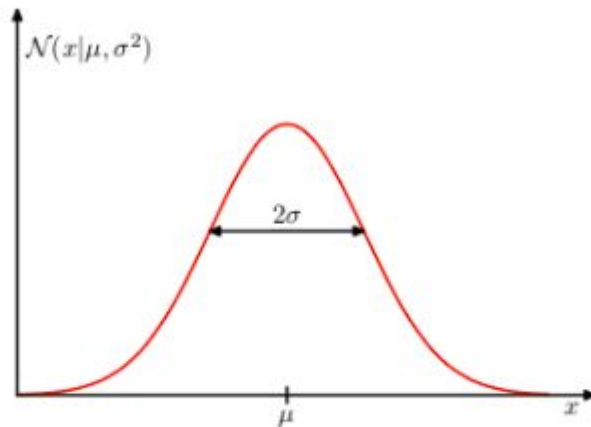
$$y(x, \mathbf{w}) = w_0 + w_1x + w_2x^2 + \dots + w_Mx^M = \sum_{j=0}^M w_jx^j$$





Regresión polinomial. Solución por MV

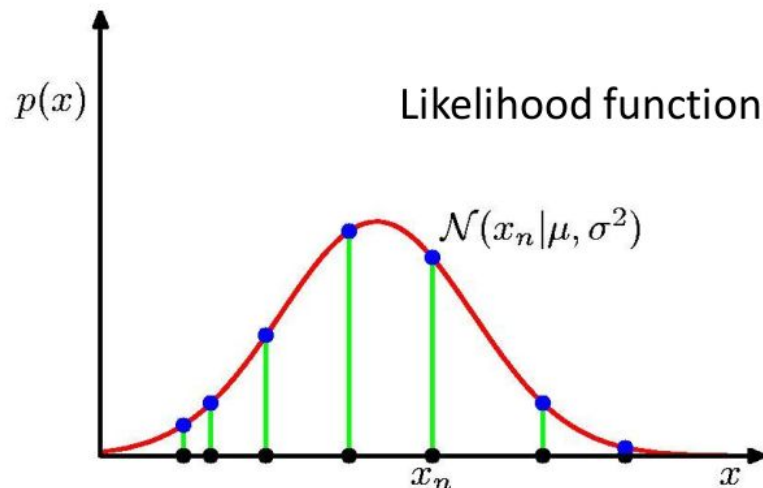
Distribución gaussiana



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

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

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 $\ln p(\mathbf{x}|\mu, \sigma^2) = -\frac{1}{2\sigma^2} \sum_{n=1}^N (x_n - \mu)^2 - \frac{N}{2} \ln \sigma^2 - \frac{N}{2} \ln(2\pi)$

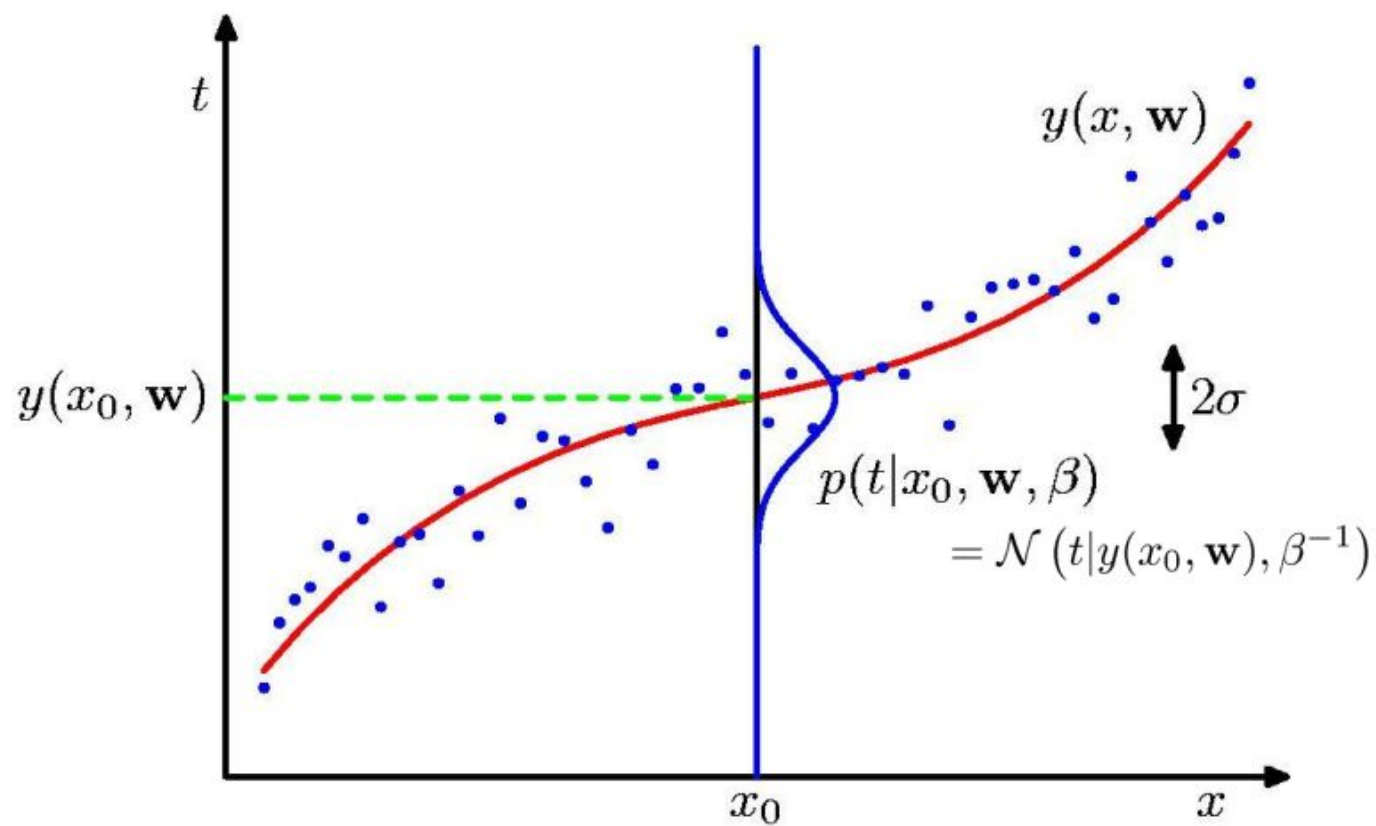
- Media muestral por MV $\mu_{ML} = \frac{1}{N} \sum_{n=1}^N x_n$

- Varianza muestral por MV $\sigma_{ML}^2 = \frac{1}{N} \sum_{n=1}^N (x_n - \mu_{ML})^2$

Revisando el ajuste de curvas

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

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



Probabilidades bayesianas

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

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

posterior \propto likelihood \times prior

- El denominador $p(D)$ es un factor de normalización

Revisando el ajuste de curvas

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

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

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

- La solución por MV, después de notar que los últimos dos términos no dependen de w y que β 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 β :

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

- Con \mathbf{w} y β podemos hacer predicciones sobre x mediante la "distribución predictiva"

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

- Si consideramos un *prior* Gaussiano sobre \mathbf{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 \propto likelihood \times 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

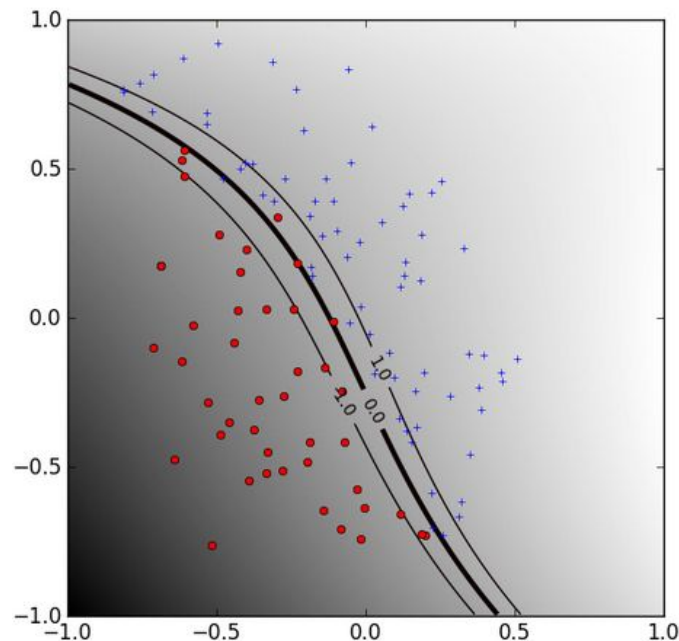
Clasificación basada en probabilidades

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

- Recordar:

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

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



Regresión lineal

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

- Ecuaciones normales

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

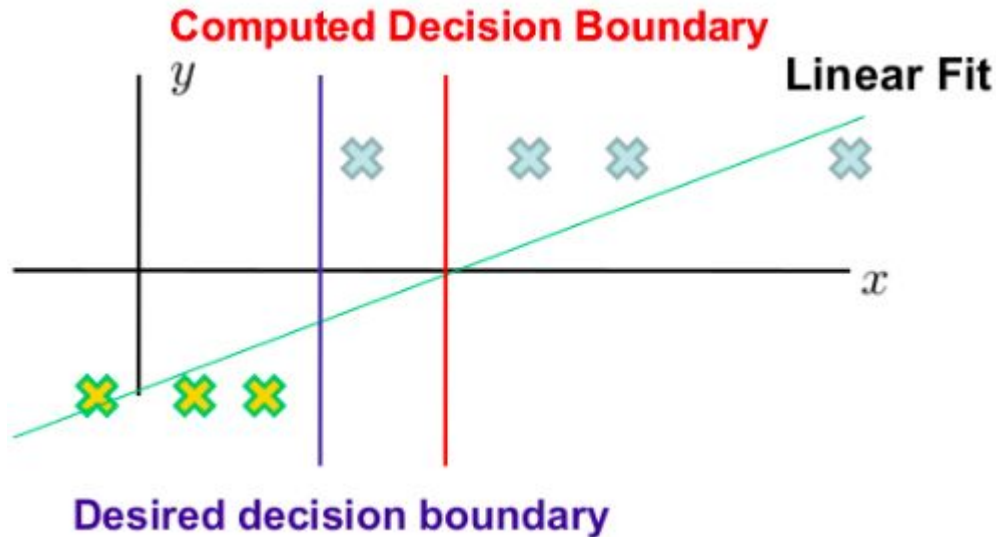
$$\mathbf{e} = \mathbf{y} - \mathbf{X}\mathbf{w}$$

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

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

Error cuadrático en clasificación

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

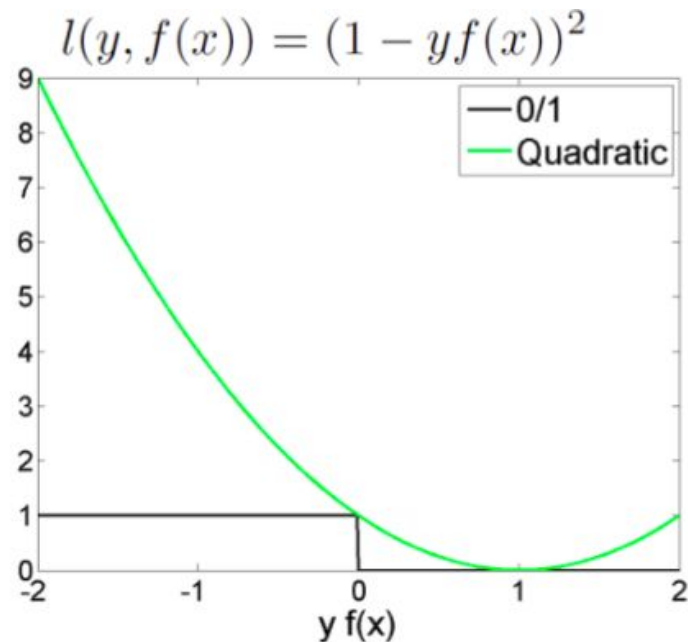


Error cuadrático en clasificación

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

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

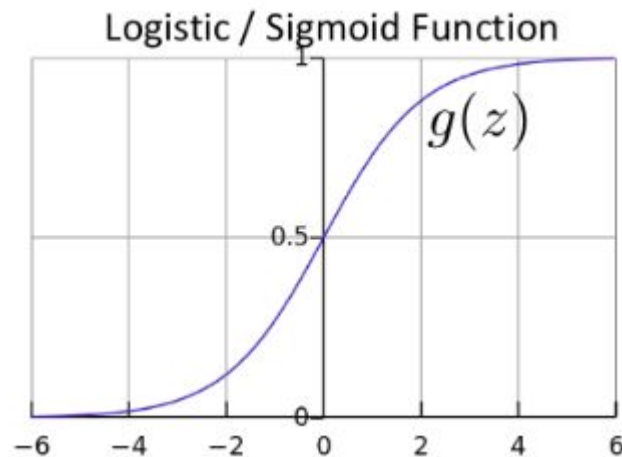
- No es robusta frente a *outliers*
- Penaliza predicciones que son buenas



Regresión logística

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

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



WARNING: draft mode begins

Interpretation of Hypothesis Output

$$h_{\theta}(\mathbf{x}) = \text{estimated } p(y = 1 \mid \mathbf{x}; \theta)$$

Example: Cancer diagnosis from tumor size

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumorSize} \end{bmatrix}$$

$$h_{\theta}(\mathbf{x}) = 0.7$$

→ Tell patient that 70% chance of tumor being malignant

Note that: $p(y = 0 \mid \mathbf{x}; \theta) + p(y = 1 \mid \mathbf{x}; \theta) = 1$

Therefore, $p(y = 0 \mid \mathbf{x}; \theta) = 1 - p(y = 1 \mid \mathbf{x}; \theta)$

Another Interpretation

- Equivalently, logistic regression assumes that

$$\log \frac{p(y = 1 \mid \mathbf{x}; \boldsymbol{\theta})}{p(y = 0 \mid \mathbf{x}; \boldsymbol{\theta})} = \theta_0 + \theta_1 x_1 + \dots + \theta_d x_d$$

odds of $y = 1$

Measure	Min	Max	Name
$\Pr(Y = 1)$	0	1	"probability"
$\frac{\Pr(Y = 1)}{1 - \Pr(Y = 1)}$	0	∞	"odds"
$\log\left(\frac{\Pr(Y = 1)}{1 - \Pr(Y = 1)}\right)$	$-\infty$	∞	"log-odds" or "logit"

Side Note: the odds in favor of an event is the quantity $p / (1 - p)$, where p is the probability of the event

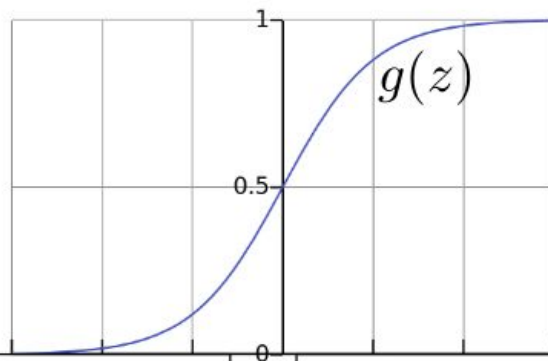
E.g., If I toss a fair dice, what are the odds that I will have a 6?

- In other words, logistic regression assumes that the log odds is a linear function of \mathbf{x}

Logistic Regression

$$h_{\theta}(x) = g(\theta^T x)$$

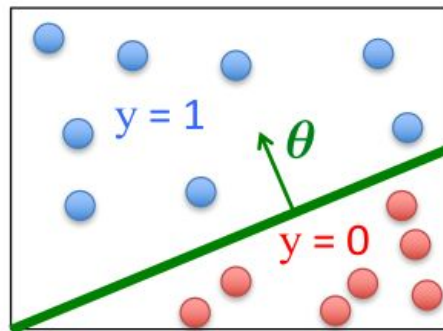
$$g(z) = \frac{1}{1 + e^{-z}}$$



$\theta^T x$ should be large negative values for negative instances

$\theta^T x$ should be large positive values for positive instances

- Assume a threshold and...
 - Predict $y = 1$ if $h_{\theta}(x) \geq 0.5$
 - Predict $y = 0$ if $h_{\theta}(x) < 0.5$



Logistic Regression

- Given $\left\{ \left(\mathbf{x}^{(1)}, y^{(1)} \right), \left(\mathbf{x}^{(2)}, y^{(2)} \right), \dots, \left(\mathbf{x}^{(n)}, y^{(n)} \right) \right\}$

where $\mathbf{x}^{(i)} \in \mathbb{R}^d$, $y^{(i)} \in \{0, 1\}$

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

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

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix}$$

$$\mathbf{x}^{\top} = \begin{bmatrix} 1 & x_1 & \dots & x_d \end{bmatrix}$$

Logistic Regression Objective Function

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

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

- Using the logistic regression model

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

results in a non-convex optimization

A probabilistic criterion for training a classifier

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

y: discrete observations: model as samples from Bernoulli distribution

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

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

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

Find \mathbf{w} that maximizes the likelihood of labels in the training set

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

Intuition Behind the Objective

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

- Cost of a single instance:

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

- Can re-write objective function as

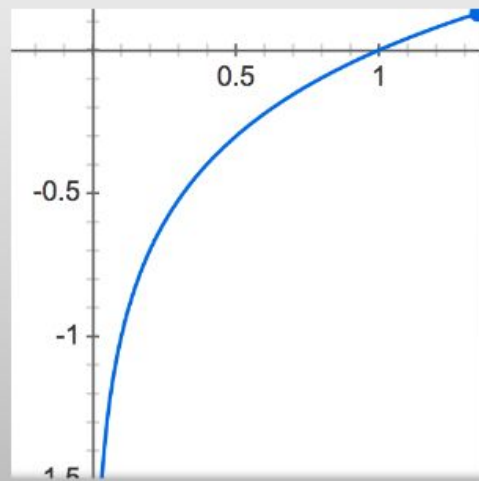
$$J(\boldsymbol{\theta}) = \sum_{i=1}^n \text{cost}(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), y^{(i)})$$

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

Intuition Behind the Objective

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

Aside: Recall the plot of $\log(z)$

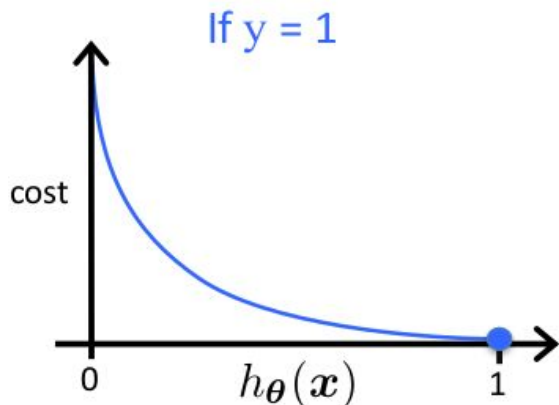


Intuition Behind the Objective

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

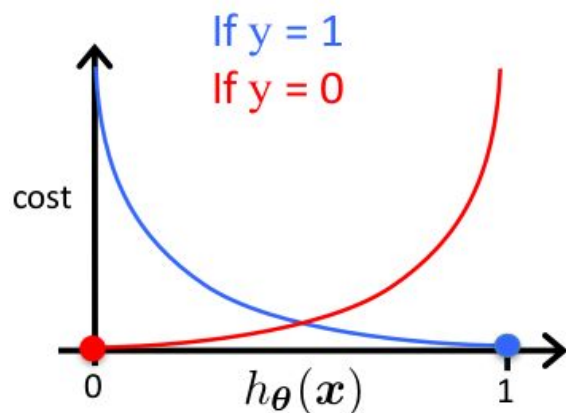
If $y = 1$

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



Intuition Behind the Objective

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



If $y = 0$

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

Regularized Logistic Regression

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

- We can regularize logistic regression exactly as before:

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

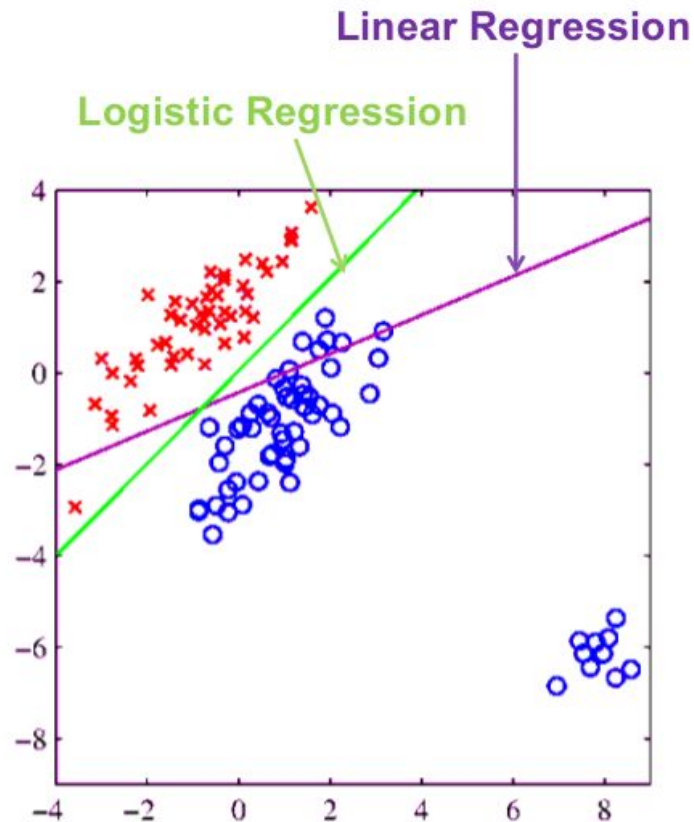
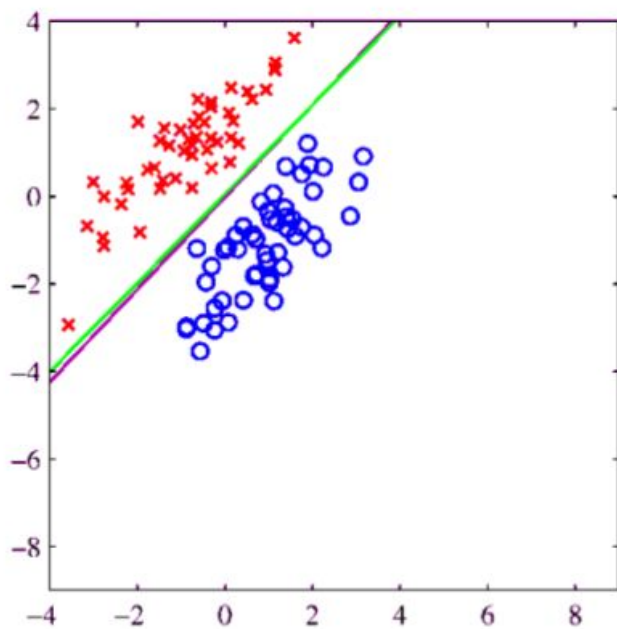


[1:d] => exclude the bias!

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

Logistic vs Linear Regression

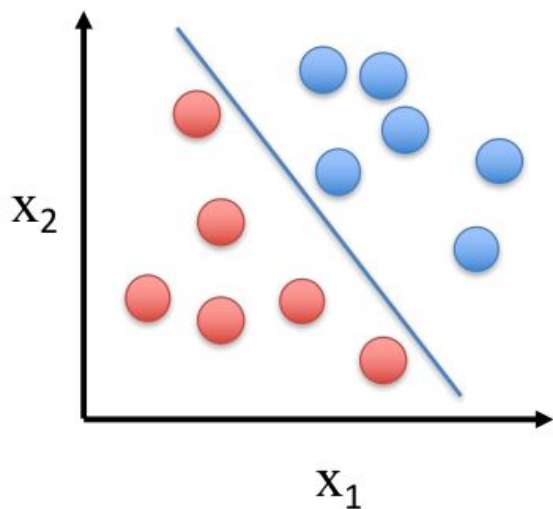
Logistic regression is more robust



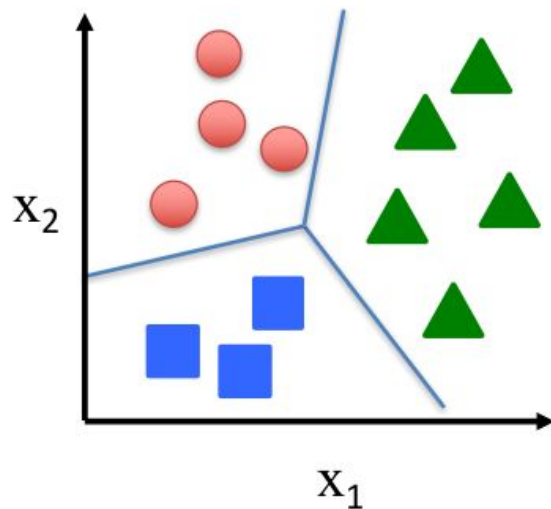
Problemas multiclase

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:

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



weight assigned to $y = 0$ weight assigned to $y = 1$

- For C classes $\{1, \dots, C\}$:

$$p(y = c \mid x; \theta_1, \dots, \theta_C) = \frac{\exp(\theta_c^T x)}{\sum_{c=1}^C \exp(\theta_c^T x)}$$

– Called the **softmax** function

Implementing Multi-Class Logistic Regression

- Use $h_c(\mathbf{x}) = \frac{\exp(\boldsymbol{\theta}_c^\top \mathbf{x})}{\sum_{c=1}^C \exp(\boldsymbol{\theta}_c^\top \mathbf{x})}$ 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

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

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 $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}$ $\begin{matrix} \mathbf{x} \in \mathbb{R}^n \\ \mathbf{y} \in \{1, 2, \dots, K\} \end{matrix}$
 - 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 $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_K$ using any learning algorithm we have seen

1. One-vs-all classification

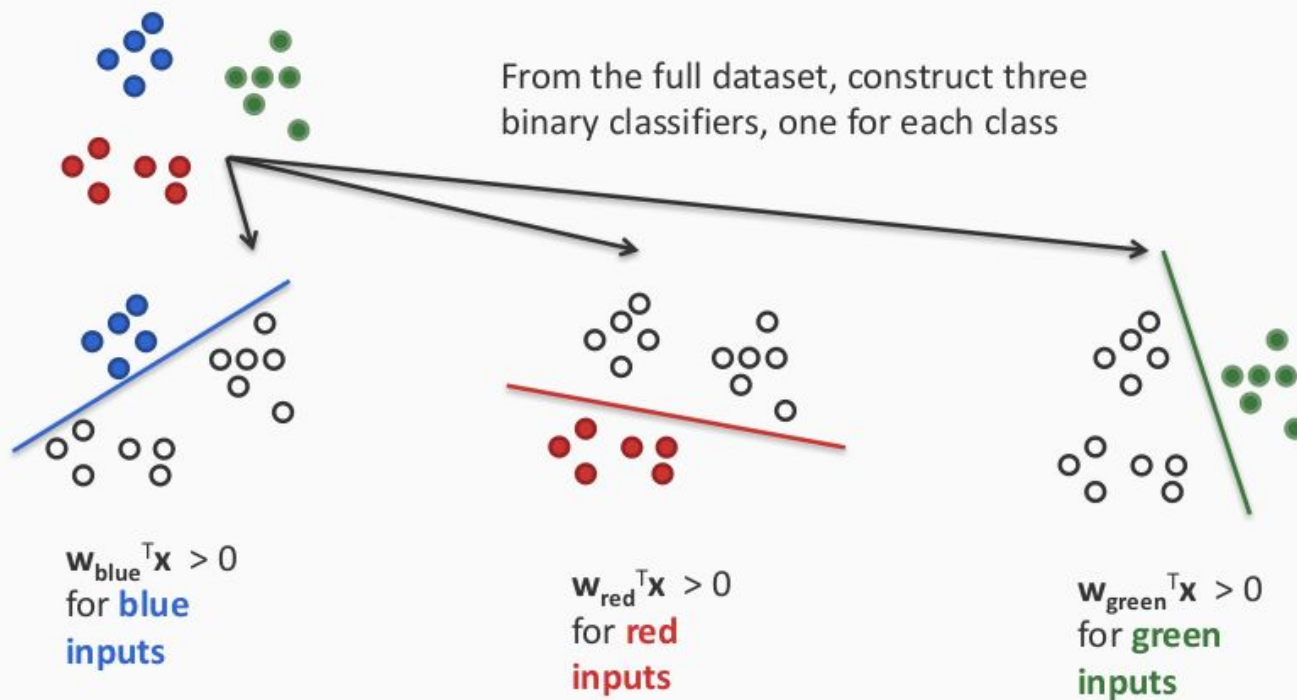
- **Assumption:** Each class individually separable from *all* the others

- **Learning:** Given a dataset $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}$ $\begin{matrix} \mathbf{x} \in \mathbb{R}^n \\ \mathbf{y} \in \{1, 2, \dots, K\} \end{matrix}$
 - Train K binary classifiers $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_K$ using any learning algorithm we have seen

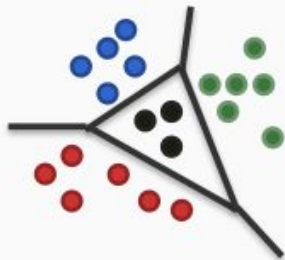
- **Prediction:** “Winner Takes All”

$$\operatorname{argmax}_i \mathbf{w}_i^\top \mathbf{x}$$

Visualizing One-vs-all



One-vs-all may not always work

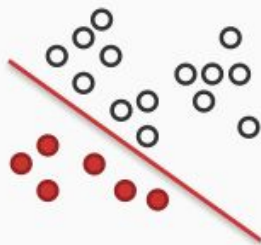


Black points are not separable with a single binary classifier

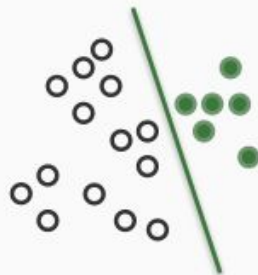
The decomposition will not work for these cases!



$w_{\text{blue}}^T x > 0$
for **blue**
inputs



$w_{\text{red}}^T x > 0$
for **red**
inputs



$w_{\text{green}}^T x > 0$
for **green**
inputs



???

2. All-vs-all classification

Sometimes called one-vs-one

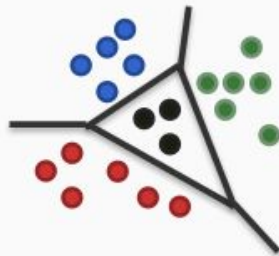
- **Assumption:** *Every* pair of classes is separable
- **Learning:** Given a dataset $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}$, $\begin{matrix} \mathbf{x} \in \mathbb{R}^n \\ \mathbf{y} \in \{1, 2, \dots, K\} \end{matrix}$
 - For every pair of labels (j , k), create a binary classifier with:
 - **Positive 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 $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}$, $\begin{matrix} \mathbf{x} \in \mathbb{R}^n \\ \mathbf{y} \in \{1, 2, \dots, K\} \end{matrix}$
 - 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

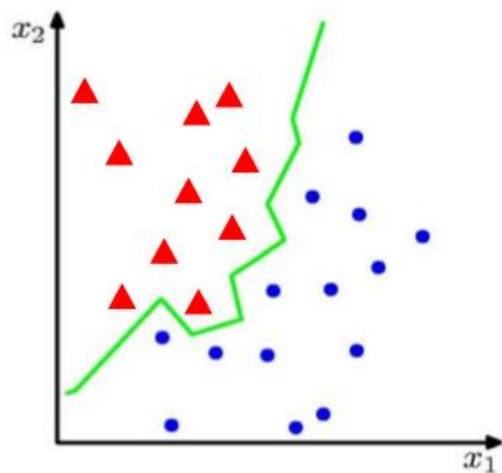
All-vs-all classification



- 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?

Modelos no paramétricos: vecinos más cercanos

Classification



- Suppose we are given a training set of N observations

(x_1, \dots, x_N) and (y_1, \dots, 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

$$f(x_i) = y_i$$

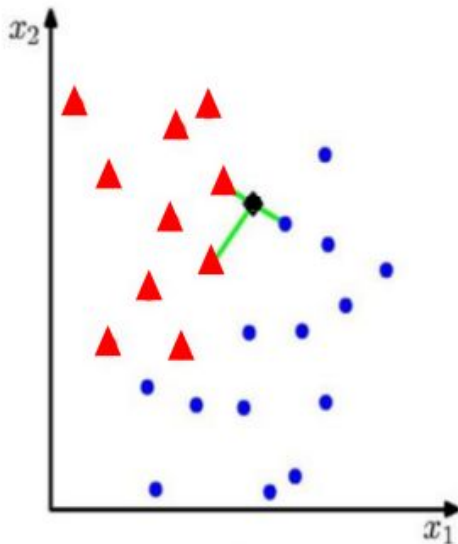
K Nearest Neighbour (K-NN) Classifier

Algorithm

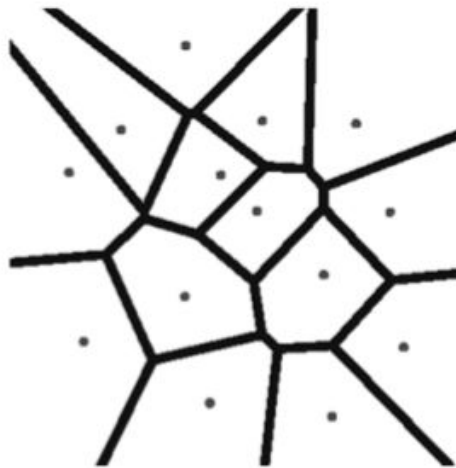
- For each test point, x , to be classified, find the K nearest samples in the training data
- Classify the point, x , according to the majority vote of their class labels

e.g. $K = 3$

- applicable to multi-class case

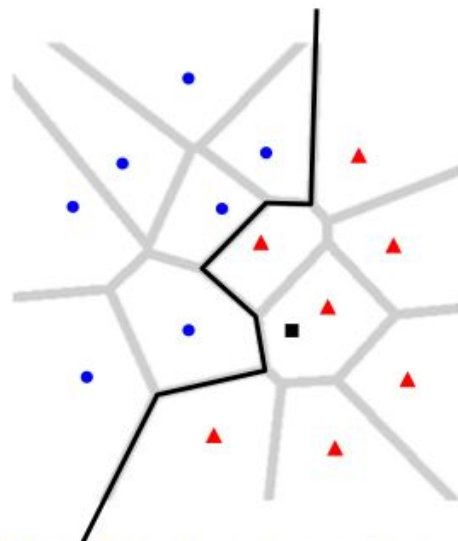


$$K = 1$$



Voronoi diagram:

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

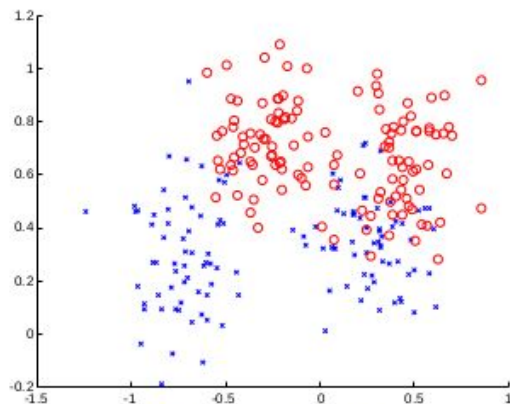


Classification boundary:

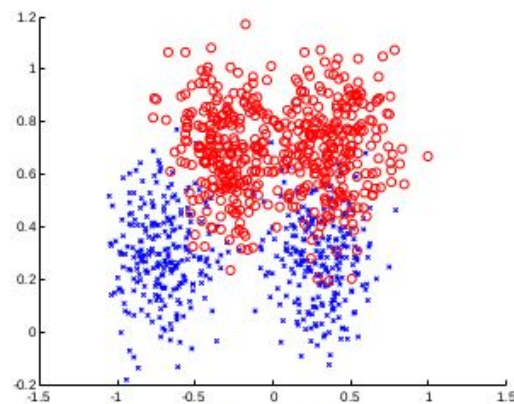
- non-linear

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 $= \frac{1}{N} \sum_{i=1}^N \underbrace{[y_i \neq f(x_i)]}_{\text{loss function}}$ The “risk”



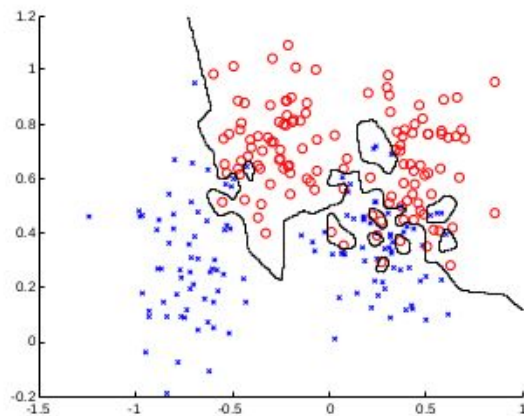
Training data



Testing data

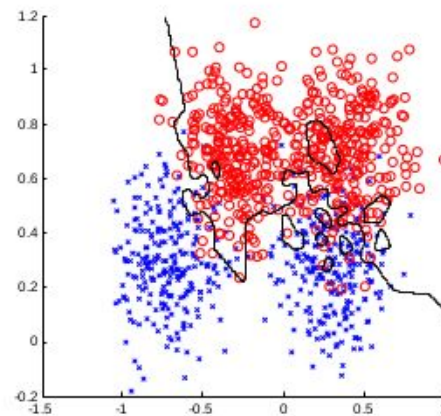
$K = 1$

Training data



error = 0.0

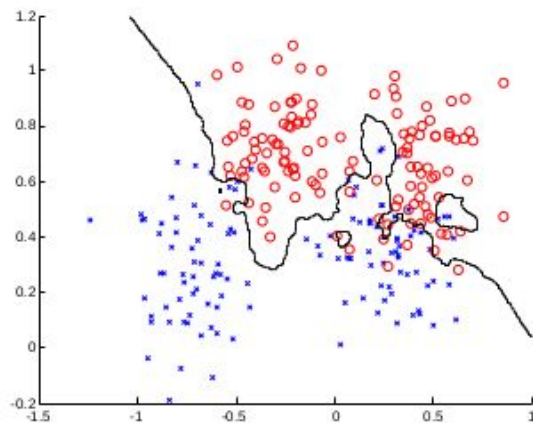
Testing data



error = 0.15

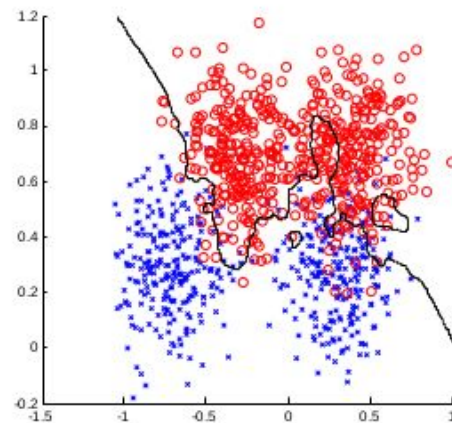
$K = 3$

Training data



error = 0.0760

Testing data



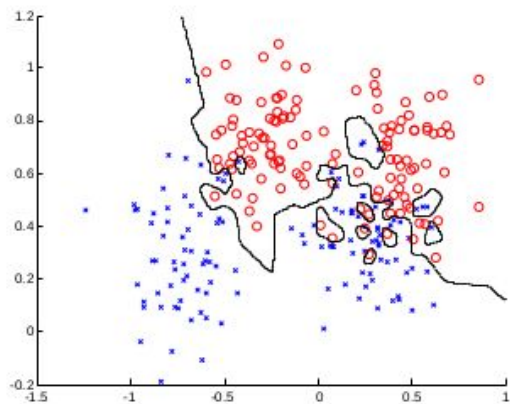
error = 0.1340

Generalization

- The real aim of supervised learning is to do well on test data that is not known during learning
- Choosing the values for the parameters that minimize the loss function on the training data is not necessarily the best policy
- We want the learning machine to model the true regularities in the data and to ignore the noise in the data.

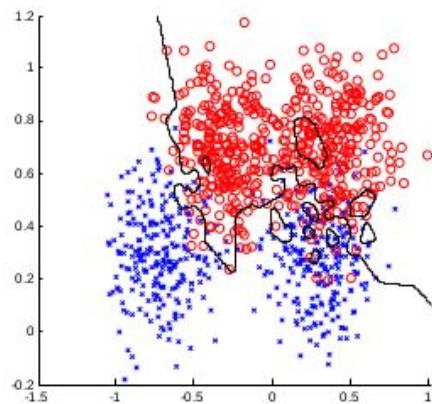
$K = 1$

Training data



error = 0.0

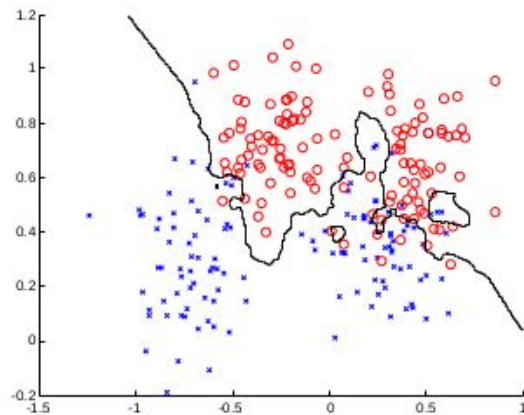
Testing data



error = 0.15

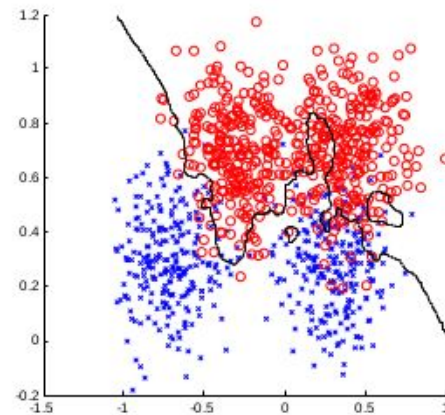
$K = 3$

Training data



error = 0.0760

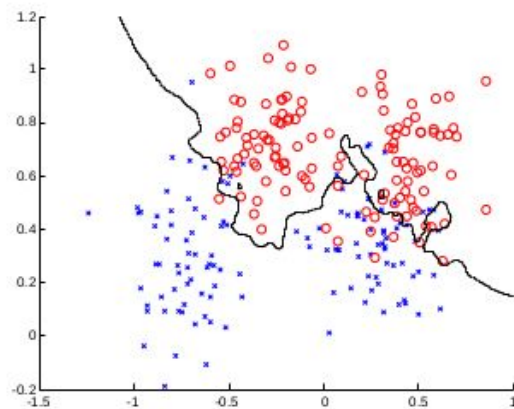
Testing data



error = 0.1340

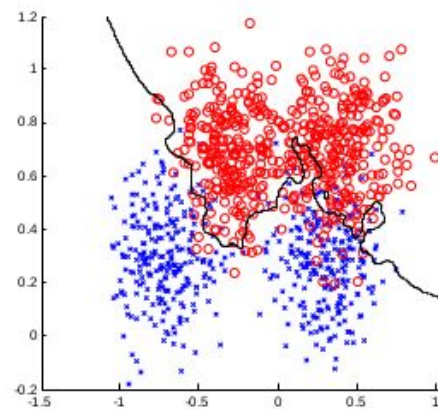
$K = 7$

Training data



error = 0.1320

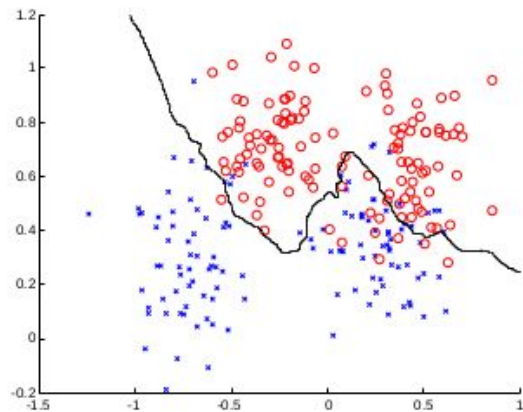
Testing data



error = 0.1110

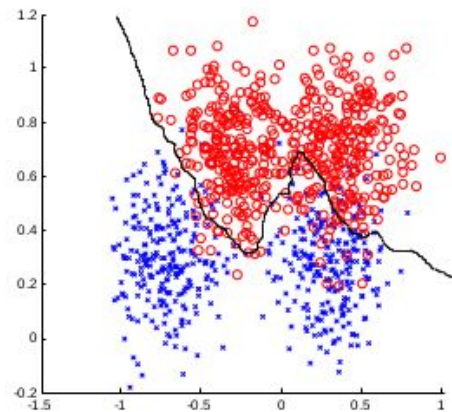
$K = 21$

Training data



error = 0.1120

Testing data



error = 0.0920

Properties and training

As K increases:

- Classification boundary becomes smoother
- Training error can increase

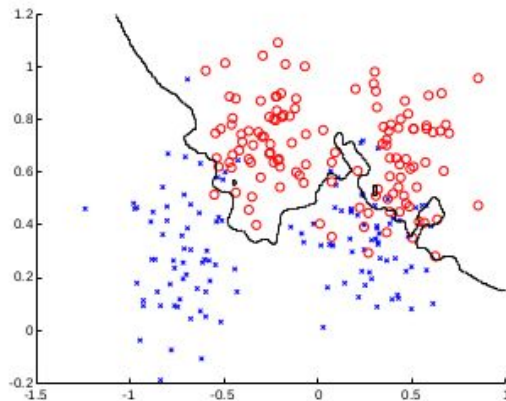
Choose (learn) K by cross-validation

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

Summary

Advantages:

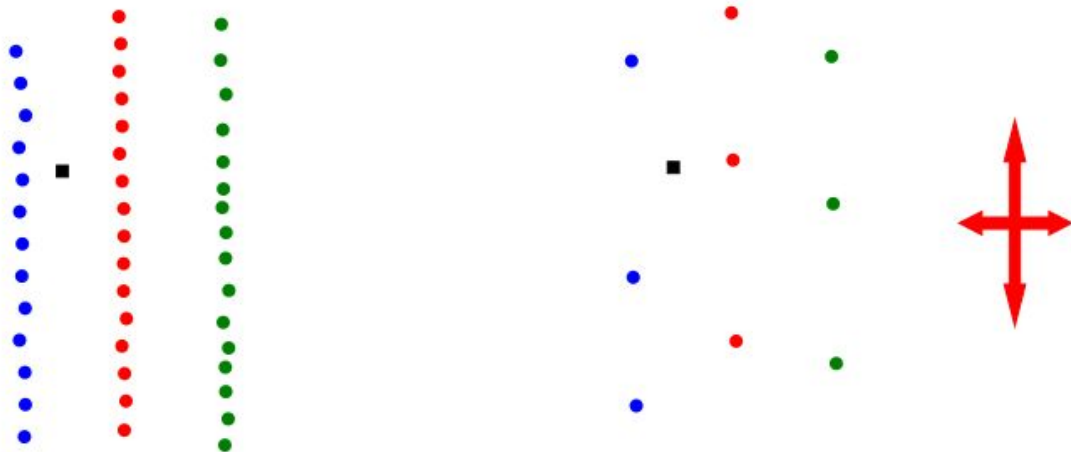
- K-NN is a simple but effective classification procedure
- Applies to multi-class classification
- Decision surfaces are non-linear
- Quality of predictions automatically improves with more training data
- Only a single parameter, K ; easily tuned by cross-validation



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 (opcional)

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

- **Learning:**
 - Represent each label by a bit string
 - Train one binary classifier for each bit

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

- **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

Answer: Use redundancy

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

#	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

- 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?

- 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
 - 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

#	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

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)