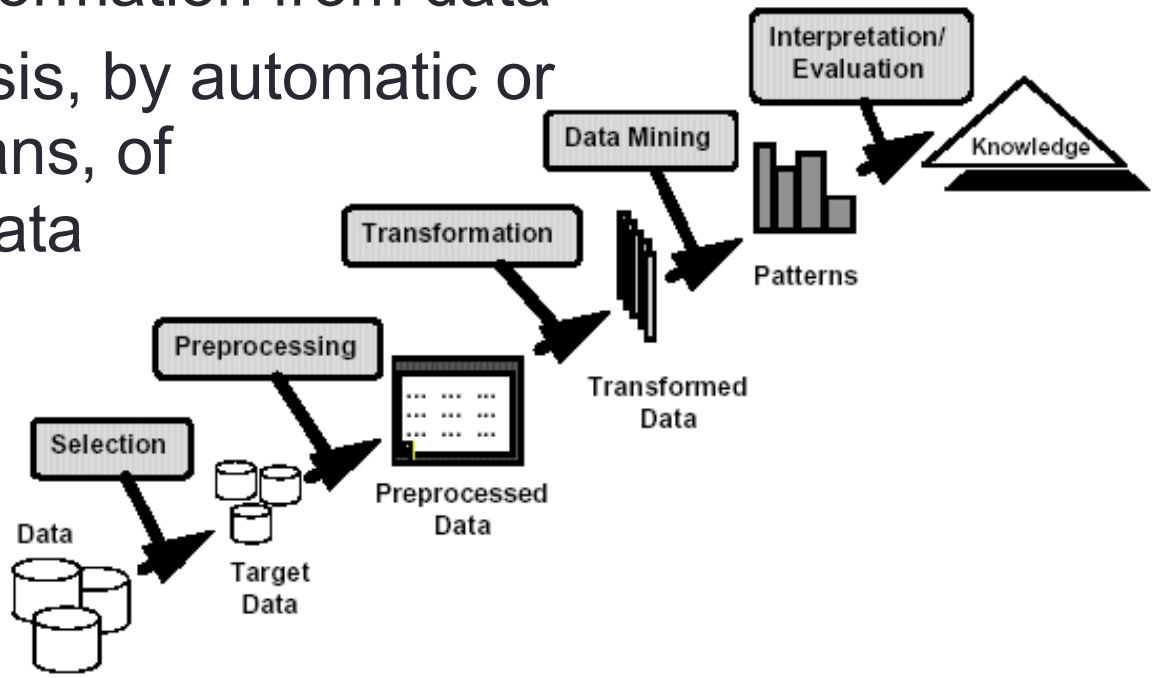
MINERÍA DE DATOS: PREPROCESAMIENTO Y CLASIFICACIÓN

Un mundo de datos

- Nuestro mundo gira cada vez mas en torno a los datos:
  - **Ciencia**: astronomía, genómica, medio-ambiente. . .
  - **Industria y Energía**: redes de sensores, gestión parques eólicos, previsión de demanda, ciudades inteligentes. . .
  - **Ciencias sociales y humanidades**: libros digitalizados, documentos históricos, datos sociales. . .
  - **Entretenimiento**: sistemas de recomendación, contenidos digitales, búsquedas multimedia. . .
  - **Medicina**: examen de imágenes medicas, previsión de demanda en hospitales, sistemas expertos. . .
  - **Financias y negocios**: transacciones de mercados automatizadas. . .

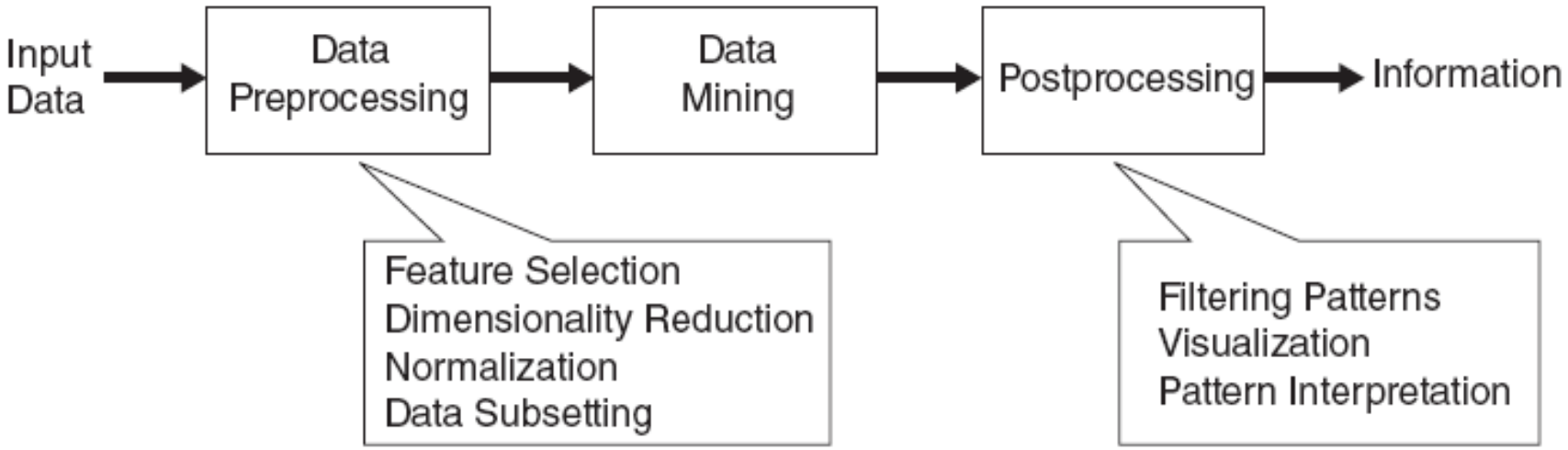
What is Data Mining?

- Many Definitions
  - Non-trivial extraction of implicit, previously unknown and potentially useful information from data
  - Exploration & analysis, by automatic or semi-automatic means, of large quantities of data in order to discover meaningful patterns



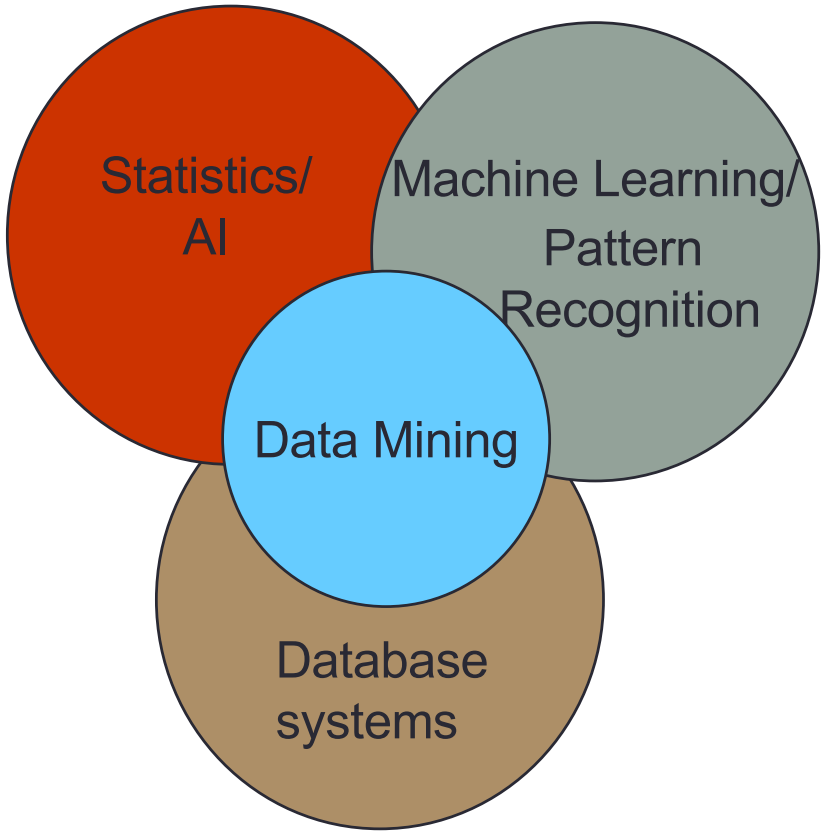
"Introduction to Data Mining", Pang-Ning Tan, Michael Steinbach, Vipin Kumar, Pearson, 2014

What is Data Mining?



Origins of Data Mining

- Draws ideas from machine learning/AI, pattern recognition, statistics, and database systems
- Traditional Techniques may be unsuitable due to
  - Enormity of data
  - High dimensionality of data
  - Heterogeneous, distributed nature of data



Data Mining Tasks

- Prediction Methods
  - Use some variables to predict unknown or future values of other variables
- Description Methods
  - Find human-interpretable patterns that describe the data

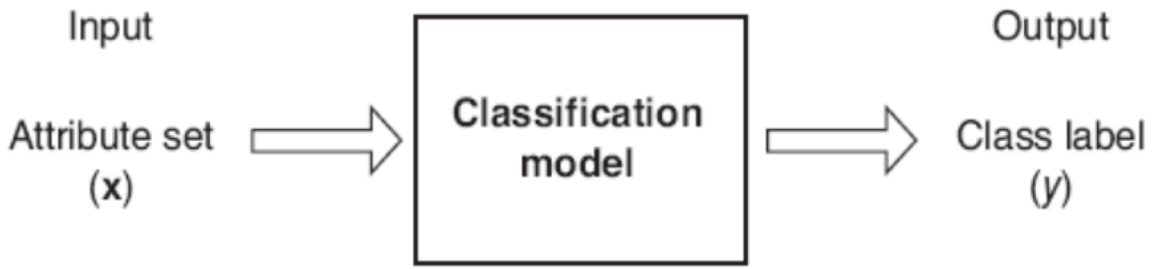
Data Mining Tasks...

- Classification [Predictive]
- Regression [Predictive]
- Clustering [Descriptive]
- Association Rule Discovery [Descriptive]
- Sequential Pattern Discovery [Descriptive]
- Deviation Detection [Predictive]

Classification: Definition

- Given a collection of records (*training set*)
  - Each record contains a set of *attributes*, one of the attributes is the *class*.
- Find a *model* for class attribute as a function of the values of other attributes.
- Goal: previously unseen records should be assigned a class as accurately as possible.
  - A *test set* is used to determine the accuracy of the model. Usually, the given data set is divided into training and test sets, with training set used to build the model and test set used to validate it.

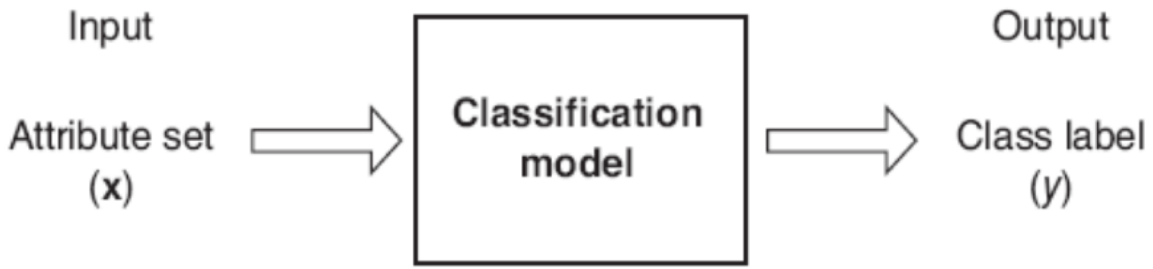
Classification: Definition



- Descriptive Modeling
- Predictive Modeling

- Accuracy
- Interpretability

Classification: Definition



- Descriptive Modeling
- Predictive Modeling

- Accuracy
- Interpretability

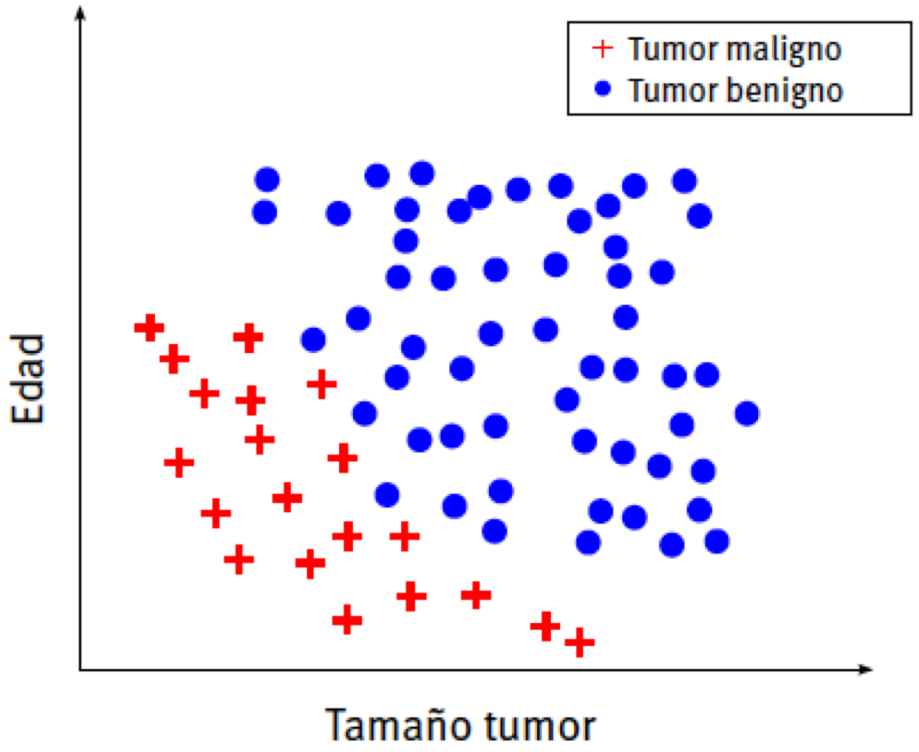
Explainable artificial intelligence (XAI)

Tipos de clasificación

- Clasificación binaria
- Clasificación multi-clase
- Clasificación ordinal
- Clasificación multi-etiqueta

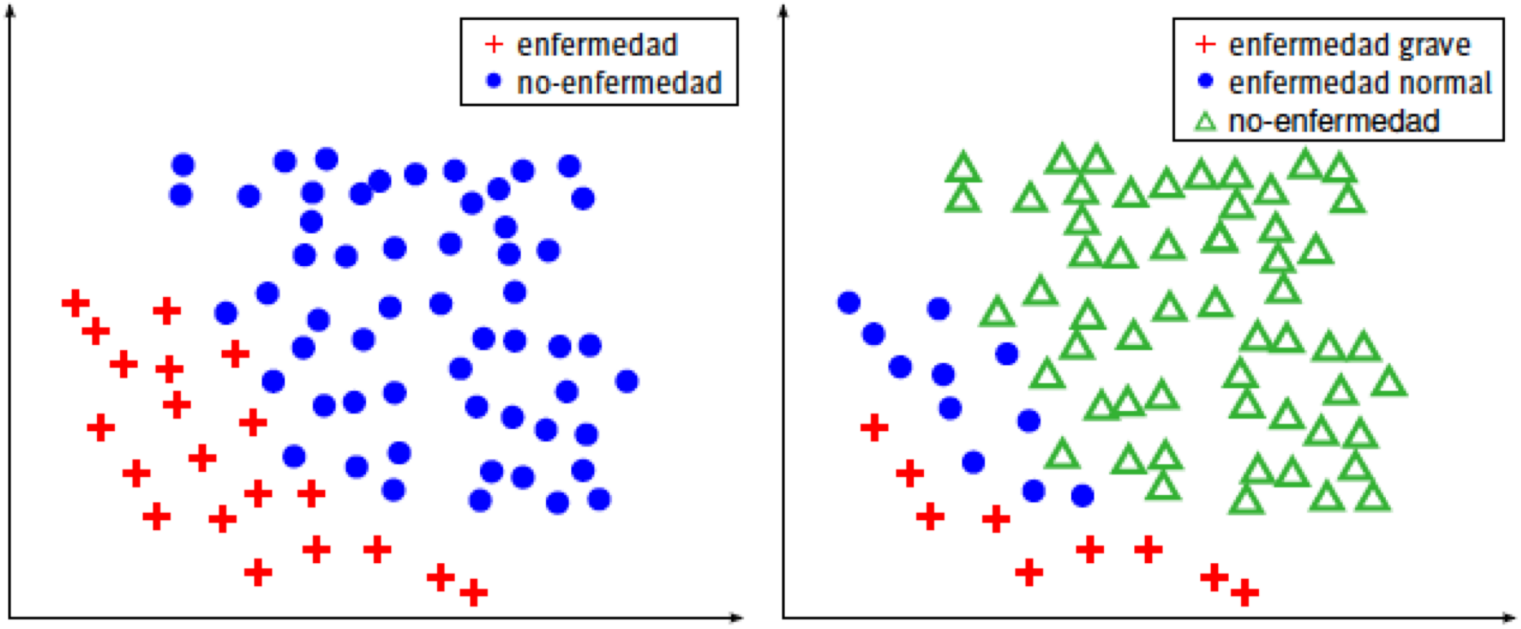
Clasificación binaria

- Ejemplo de problema de clasificación ¿Podemos estimar un diagnostico basado en el tamaño del tumor y la edad del paciente?



Clasificación multi-clase

- Un ejemplo de clasificación binaria (figura a la izquierda) frente a la clasificación multi-clase (figura de la derecha). En el primer caso hay dos estados para un patrón: enfermo o no enfermo. Sin embargo, un experto que se apoye en técnicas de aprendizaje automático puede demandar grados de clasificación, en cuyo caso podrá afrontarse el problema como clasificación multi-clase.



Clasificación ordinal

- Clasificación nominal, pero con con un orden asociado entre las clases.
- Por ejemplo, el riesgo crediticio asociado con las obligaciones financieras de una empresa en el que, por ejemplo, si consideramos el ratio de crédito a largo plazo de S&P encontramos las clases

(AAA,AA+,AA,AA-,A+, ...,C,RD,SD,D).

- En este ejemplo, si consideramos que la clasificación actual de una empresa es AA y un algoritmo lo clasifica como AA- y otro algoritmo lo clasifica como D, ambos algoritmos han realizado una clasificación errónea, pero el error en el segundo algoritmo es mayor que en el primero, lo que podría acarrear decisiones problemáticas en la empresa.

Clasificación multi-etiqueta

¿Contiene una casa? Si o No

¿Qué etiquetas son relevantes para esta imagen?

Casa: Si
Árbol: Si
Playa: No
Nubes: Si
Montañas: No
Animales: No

Este tipo de problemas, en los que tenemos un conjunto de variables objetivo, se conocen como problemas de clasificación multi-etiqueta.

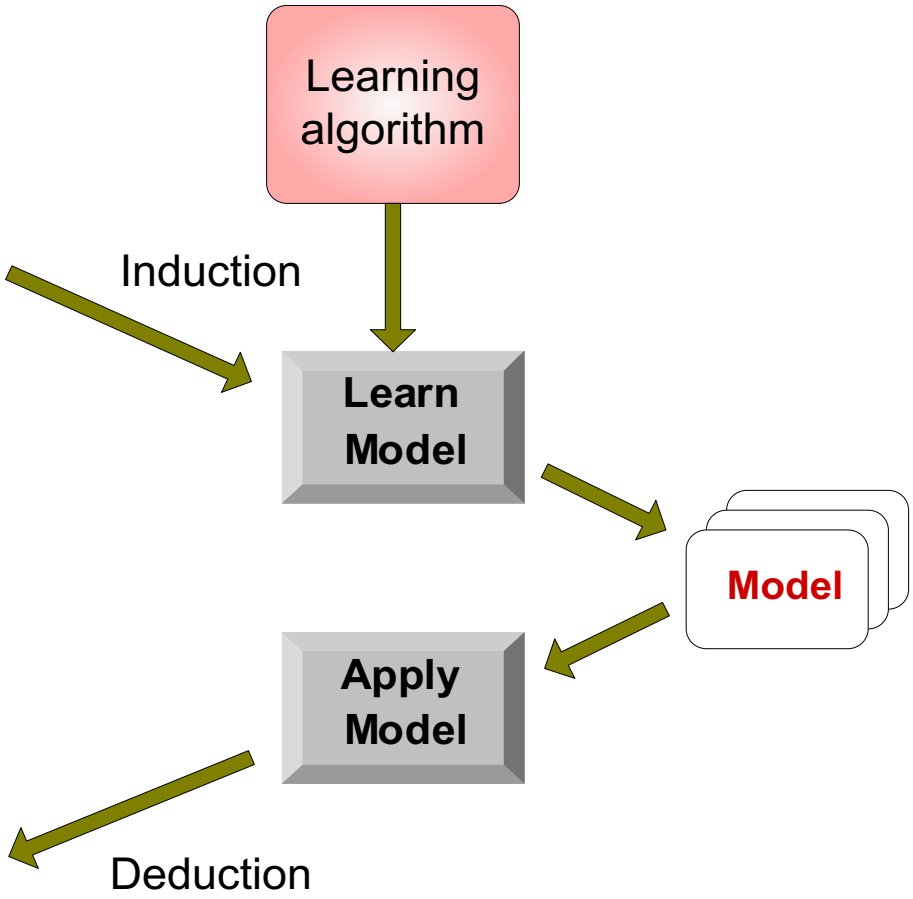
Illustrating Classification Task

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set



Ejemplo de problema de clasificación

- Ejemplo: el problema de clasificación de la flor de lirios (IRIS)
- Tres clases de lirios: setosa, versicolor y virginica
- Cuatro atributos: longitud y anchura del pétalo y sépalo respectivamente.
- 150 ejemplos, 50 de cada clase (disponible en UC Irvine Machine Learning Repository http://archive.ics.uci.edu/ml/index.php)

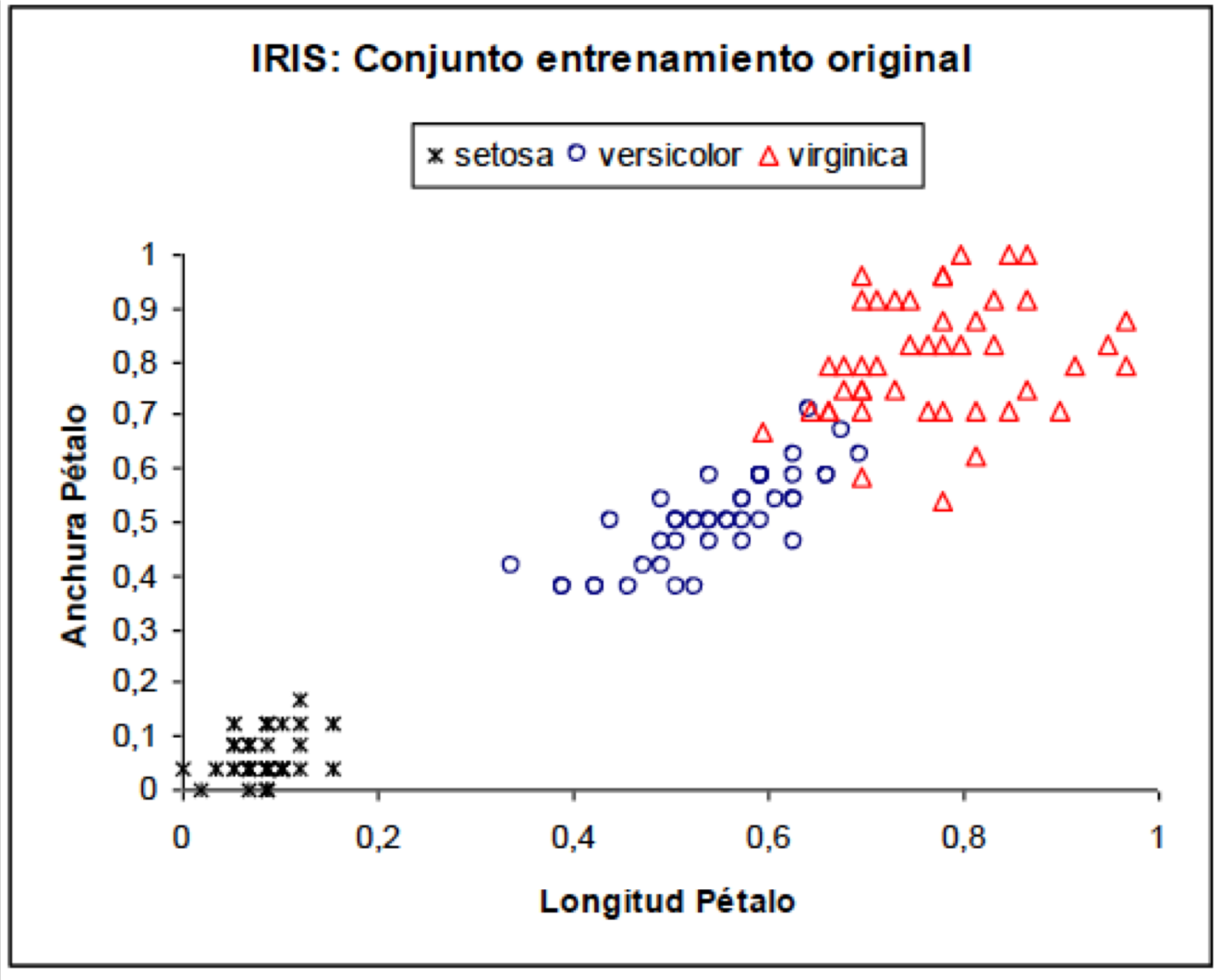
	Sepal length X_1	Sepal width X_2	Petal length X_3	Petal width X_4	Class X_5
$\mathbf{x}_1$	5.9	3.0	4.2	1.5	Iris-versicolor
$\mathbf{x}_2$	6.9	3.1	4.9	1.5	Iris-versicolor
$\mathbf{x}_3$	6.6	2.9	4.6	1.3	Iris-versicolor
$\mathbf{x}_4$	4.6	3.2	1.4	0.2	Iris-setosa
$\mathbf{x}_5$	6.0	2.2	4.0	1.0	Iris-versicolor
$\mathbf{x}_6$	4.7	3.2	1.3	0.2	Iris-setosa
$\mathbf{x}_7$	6.5	3.0	5.8	2.2	Iris-virginica
$\mathbf{x}_8$	5.8	2.7	5.1	1.9	Iris-virginica
⋮	⋮	⋮	⋮	⋮	⋮
$\mathbf{x}_{149}$	7.7	3.8	6.7	2.2	Iris-virginica
$\mathbf{x}_{150}$	5.1	3.4	1.5	0.2	Iris-setosa


Setosa


Versicolor

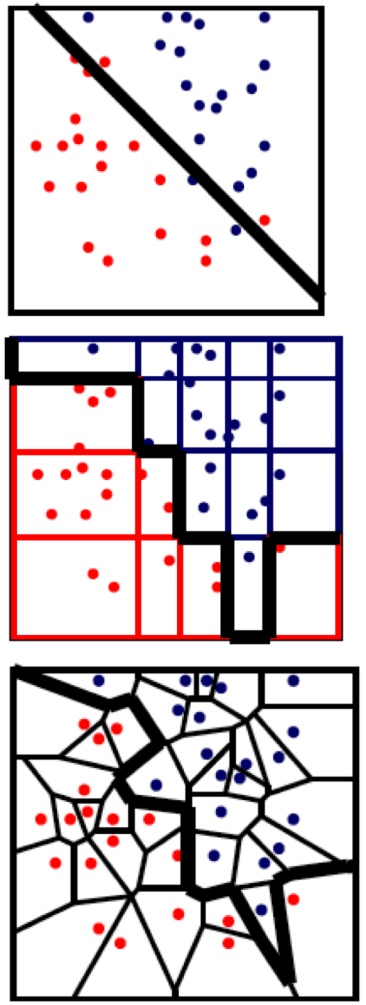

Virginica

Ejemplo de problema de clasificación



Classification Techniques

- Nonlinear models
- Decision Tree based Methods
- Rule-based Methods
- Support Vector Machines
- …



Descripción

- **Temario**:
  - Tema 1. Modelos no lineales, lineales avanzados y redes neuronales.
  - Tema 2. Árboles de decisión. Aprendizaje de Reglas.
  - Tema 3 y 4. Multiclasificación, Ensembles y Descomposición de problemas multiclase.
  - Tema 5. Máquinas soporte vectorial (SVM).
  - Tema 6. Preprocesamiento de datos.

- **Bibliografía**:

  - “The Elements of Statistical Learning: Data Mining, Inference, and Prediction”, Trevor Hastie, Robert Tibshirani, Jerome Friedman. Second Edition, Springer, 2009.
  - "Data Preparation for Data Mining". Dorian Pyle. Morgan Kaufmann, 1999.
  - "Data Preprocessing in Data Mining". Salvador García, Julián Luengo, Francisco Herrera. Springer, 2015.
  - "Data Mining: Concepts and Techniques. Jiawei Han, Jian Pei, Micheline Kamber, 2011.
  - “Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems”. Aurélien Géron. O’Reilly, 2019.
  - “Hands-On Data Preprocessing in Python: Learn how to effectively prepare data for successful data analytics”. Roy Jafair. Packt, 2022.

Descripción

- **Temario**:
  - Tema 1. Modelos no lineales, lineales avanzados y redes neuronales.
  - Tema 2. Árboles de decisión. Aprendizaje de Reglas.
  - Tema 3 y 4. Multiclasificación, Ensembles y Descomposición de problemas multiclase.
  - Tema 5. Máquinas soporte vectorial (SVM).
  - Tema 6. Preprocesamiento de datos.

- **Bibliografía**:
  - ”The Elements of Statistical Learning: Data Mining, Inference, and Prediction”, Trevor Hastie, Robert Tibshirani, Jerome Friedman. Second Edition, Springer, 2009.
  - "Data Preparation for Data Mining". Dorian Pyle. Morgan Kaufmann, 1999.
  - "Data Preprocessing in Data Mining". Salvador García, Julián Luengo, Francisco Herrera. Springer, 2015.
  - "Data Mining: Concepts and Techniques. Jiawei Han, Jian Pei, Micheline Kamber, 2011.
  - ”Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems”. Aurélien Géron. O’Reilly, 2019.
  - “Hands-On Data Preprocessing in Python: Learn how to effectively prepare data for successful data analytics”. Roy Jafair. Packt, 2022.

NAÏVE BAYES

Minería de Datos: Preprocesamiento y clasificación

Clasificadores basados en métodos bayesianos

- Los métodos probabilísticos/bayesianos representan la incertidumbre asociada a los procesos de forma natural

  →Una gran ventaja sobre otros métodos

  Ejemplo: Supongamos que consultamos a un sistema de recomendaciones (SR) para invertir en bolsa sobre los productos P1 y P2

  - Si el modelo utilizado por el SR no trata la incertidumbre (p.e. un árbol de decisión), podríamos obtener:

    (P1, invertir) (P2, invertir)

    por lo que podríamos diversificar la cantidad a invertir entre los productos
  - Si el modelo utilizado por el SR trata la incertidumbre (p.e. una red bayesiana) podríamos obtener:

    (P1, invertir, prob=0.9) (P2, invertir, prob=0.52)

    (P1, no invertir, prob=0.1) (P2, no invertir, prob=0.48)

    por lo que repartir (al menos a partes iguales) la inversión entre ambos productos no parece lo más razonable

Teorema de Bayes e hipótesis MAP

- El teorema de Bayes orientado a un problema de clasificación con n variables tiene la siguiente expresión

$$P(C|A_1,\ldots,A_n) = \frac{P(A_1,\ldots,A_n|C)P(C)}{P(A_1,...,A_n)}$$

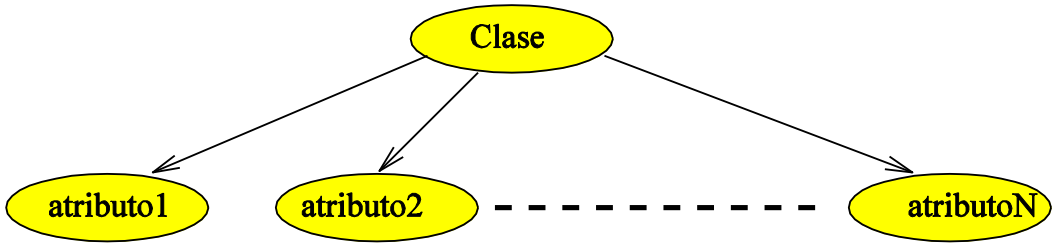
- Hipótesis MAP (máxima a posteriori): Si queremos clasificar un nuevo caso $(a_1,\ldots a_n)$ y la variable clase C tiene k posibles categorías $\Omega_C=\{c_1,\ldots,c_k\}$, lo que nos interesa es identificar la más probable y devolverla como clasificación

$$\begin{aligned} c_{MAP} &= \arg\max_{c\in\Omega_C} P(c|a_1,\ldots,a_n) \\ &= \arg\max_{c\in\Omega_C} \frac{P(a_1,\ldots,a_n|c)P(c)}{P(a_1,\ldots,a_n)} \\ &= \arg\max_{c\in\Omega_C} P(a_1,\ldots,a_n|c)P(c) \end{aligned}$$

- Problema: Hay que trabajar con la distribución conjunta y eso normalmente es inmanejable

Clasificador Naïve Bayes

- Es el modelo de red bayesiana orientada a clasificación más simple
- Supone que todos los atributos son independientes conocida la variable clase. Gráficamente



- En un Naïve BAyes (NB) la hipótesis MAP queda como:

$$c_{MAP} = \arg\max_{c \in \Omega_C} P(c|a_1, \ldots, a_n) = \arg\max_{c \in \Omega_C} P(c) \prod_{i=1}^{n} P(a_i|c)$$

- A pesar de la suposición poco realista realizada en el NB, este algoritmo se considera un estándar y sus resultados son competitivos con la mayoría de los clasificadores

Clasificador Naïve Bayes

¿Cómo se estiman estas probabilidades? Estimación de párametros

- Variables discretas
  - $P(x|c_i)$ se estima como la frecuencia relativa de ejemplos que teniendo un determinado valor de x pertenecen a la clase c_i
  - Estimación por máxima verisimilitud (EMV). Sea $n(x_i)$ el nº de veces que aparece $X_i=x_i$ en la BD y $n(x_i,x_j)$ el nº de veces que aparece el par $(X_i=x_i,X_j=x_j)$ en la BD, entonces

$$p(x_i|x_j) = \frac{n(x_i,x_j)}{n(x_j)}$$

  - Suavizando por la corrección de Laplace:

$$p(x_i|x_j) = \frac{n(x_i,x_j)+1}{n(x_j)+|\Omega_{X_i}|}$$

Cuando hay muchos casos, tiende a la EMV, si hay pocos casos tiende a la uniforme

Clasificador Naïve Bayes

Ejemplo:

Day	Outlook	Temperature	Humidity	Wind	*Play Tennis?*
1	Sunny	85	85	Light	No
2	Sunny	80	90	Strong	No
3	Overcast	83	86	Light	Yes
4	Rain	70	96	Light	Yes
5	Rain	68	80	Light	Yes
6	Rain	65	70	Strong	No
7	Overcast	64	65	Strong	Yes
8	Sunny	72	95	Light	No
9	Sunny	69	70	Light	Yes
10	Rain	75	80	Light	Yes
11	Sunny	75	70	Strong	Yes
12	Overcast	72	90	Strong	Yes
13	Overcast	81	75	Light	Yes
14	Rain	71	91	Strong	No

- Estimación para *PLAY, outlook y windy*

PLAY	EMV	Lapl.
no	5/14	6/16
yes	9/14	10/16

outlook	EMV		Lapl.	
	no	yes	no	Yes
sunny	3/5	2/9	4/8	3/12
overcast	0	4/9	1/8	5/12
rainy	2/5	3/9	3/8	4/12

windy	EMV		Lapl.	
	no	yes	no	Yes
true	3/5	3/9	4/7	4/11
false	2/5	6/9	3/7	7/11

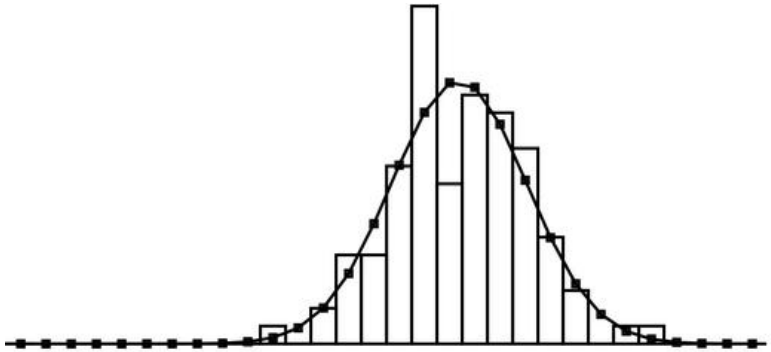
Clasificador Naïve Bayes

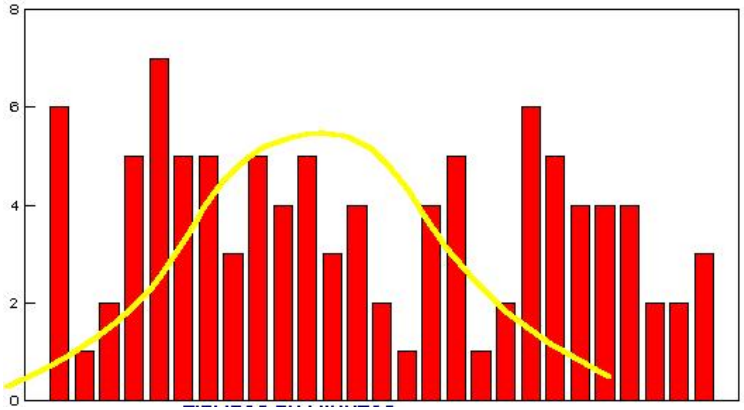
- Variables numéricas
  - $P(x|c_i)$ se estima mediante una función de densidad gaussiana. Es decir, se asume que los valores numéricos siguen una distribución normal

$$P(x|c_i) = N(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{(x-\mu)^2}{2\sigma^2}}$$

    es decir, para cada categoría de la variable clase se estima una distribución normal (de media μ y desviación estándar σ)

  - Evidentemente, en unos casos la aproximación realizada será mejor que en otros





Clasificador Naïve Bayes

Ejemplo:

Day	Outlook	Temperature	Humidity	Wind	*Play Tennis?*
1	Sunny	85	85	Light	No
2	Sunny	80	90	Strong	No
3	Overcast	83	86	Light	Yes
4	Rain	70	96	Light	Yes
5	Rain	68	80	Light	Yes
6	Rain	65	70	Strong	No
7	Overcast	64	65	Strong	Yes
8	Sunny	72	95	Light	No
9	Sunny	69	70	Light	Yes
10	Rain	75	80	Light	Yes
11	Sunny	75	70	Strong	Yes
12	Overcast	72	90	Strong	Yes
13	Overcast	81	75	Light	Yes
14	Rain	71	91	Strong	No

- Estimación para *temperature* y *humidity*
  - temperature=(PLAY=no): μ=74.6; σ=7.89
  - temperature=(PLAY=yes): μ=73; σ=6.16
  - humidity=(PLAY=no): μ=86.6; σ=9.73
  - humidity=(PLAY=yes): μ=79.11; σ=10.21

Clasificador Naïve Bayes

- Supongamos que queremos clasificar un nuevo caso:

  x=(outlook=sunny, temp=87*, hum*=90*, windy =false*)
- Suponemos la estimación por máxima verisimilitud. Aplicamos para ambas categorías de la variable clase:
  - PLAY=no

$P(x|NO) = P(NO)\cdot P_O(sunny|NO)\cdot P_w(false|NO)\cdot P_t(87|NO)\cdot P_h(90|NO)$

$= 0.36\cdot 0.6\cdot 0.4\cdot N_{temp}(87;74.6,7.89)\cdot N_{hum}(90;86.6,9.73)$

$= 0.36\cdot 0.6\cdot 0.4\cdot 0.014706\cdot 0.038573 = 4.08e\text{-}05$

  - PLAY=yes

$P(x|YES) = P(NO)\cdot P_O(sunny|YES)\cdot P_w(false|YES)\cdot P_t(87|YES)\cdot P_h(90|YES)$

$= 0.64\cdot 0.22\cdot 0.67\cdot Ntemp(87;73,6.16)\cdot Nhum(90;79.11,10.21)$

$=0.64\cdot 0.22\cdot 0.67\cdot 0.004894\cdot 0.022123 = 1.02e\text{-}05$
- Normalizando:

P(NO)=0.8 P(YES) = 0.2

Clasificador Naïve Bayes

```
# Assigning features and label variables
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','S
'Rainy','Sunny','Overcast','Overcast','Rainy']
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','

play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','
```

```
# Import LabelEncoder
from sklearn import preprocessing
#creating labelEncoder
le = preprocessing.LabelEncoder()
# Converting string labels into numbers.
wheather_encoded=le.fit_transform(wheather)
```

```
# Converting string labels into numbers
temp_encoded=le.fit_transform(temp)
label=le.fit_transform(play)
```

```
#Combinig weather and temp into single listof tuples
features=zip(weather_encoded,temp_encoded)
```

```
#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB

#Create a Gaussian Classifier
model = GaussianNB()

# Train the model using the training sets
model.fit(features,label)

#Predict Output
predicted= model.predict([[0,2]]) # 0:Overcast, 2:Mild
print "Predicted Value:", predicted
```

Clasificador Naïve Bayes

- Ventajas:
  - Es fácil de implementar
  - Obtiene buenos resultados en gran parte de los casos

- Desventajas:
  - Asumir que las variables tienen independencia condicional respecto a la clase lleva a una falta de precisión
  - En la práctica, existen dependencias entre las variables. P.e.: en datos hospitalarios:
    - Perfil: edad, historia familiar, etc.
    - Síntomas: fiebre, tos, etc.
    - Enfermedad: cáncer de pulmón, diabetes, etc.
  - Con un clasificador Naïve Bayes no se pueden modelar estas dependencias
  - Solución: Redes de creencia bayesianas, que combinan razonamiento bayesiano con relaciones causales entre los atributos

MOVING BEYOND LINEARITY

Minería de Datos: Preprocesamiento y clasificación

The truth is never linear!

Or almost never!

But often the linearity assumption is good enough.

When its not …

- ✓ polynomials,
- ✓ step functions,
- ✓ splines,
- ✓ local regression, and
- ✓ generalized additive models

offer a lot of flexibility, without losing the ease and interpretability of linear models.

Logistic regression

- Standard lineal model

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

- The output variable is discrete rather than continuous.
- Special case with binary outcomes (snows in Granada on a given day).
- How could we model and analyze such data?
  - Simply guessing “yes” or “not” is pretty crude especially if there is no perfect rule.
  - We need to fit better a stochastic model P(Y|X).
  - There’s a 25% chance of snow.

Logistic regression

- We have a binary output variable Y, and we want to model the conditional probability

$$P(Y = 1|X = x)$$

as a function of x.

Logistic regression

How can we use linear regression to solve this?

- The most obvious idea is to let p() be a linear function of x.
- The conceptual problem here is that p must be between 0 and 1, and linear functions are unbounded.
- Moreover, in many situations we empirically see that changing p by the same amount requires a bigger change in x when p is already large (or small) than when p is close to 1/2. Linear models can't do this.
- An interesting proposal has been the use of the logistic (or logit) transformation

$$\log \frac{p}{1-p}$$

- This last alternative is logistic regression.

Logistic regression

Formally, the model logistic regression model is that

$$\log \frac{p(x)}{1-p(x)} = \beta_0 + x \cdot \beta$$

Solving for p, this gives

$$p(x; b, w) = \frac{e^{\beta_0 + x \cdot \beta}}{1 + e^{\beta_0 + x \cdot \beta}} = \frac{1}{1 + e^{-(\beta_0 + x \cdot \beta)}}$$

- We predict Y = 1 when p ≥ 0.5 and Y = 0 when p < 0.5.

Logistic regression

With several antecedent variables:

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_P x_P.$$

$$p = \frac{1}{1 + \exp\left[-(\beta_0 + \beta_1 x_1 + \cdots + \beta_P x_P)\right]}$$

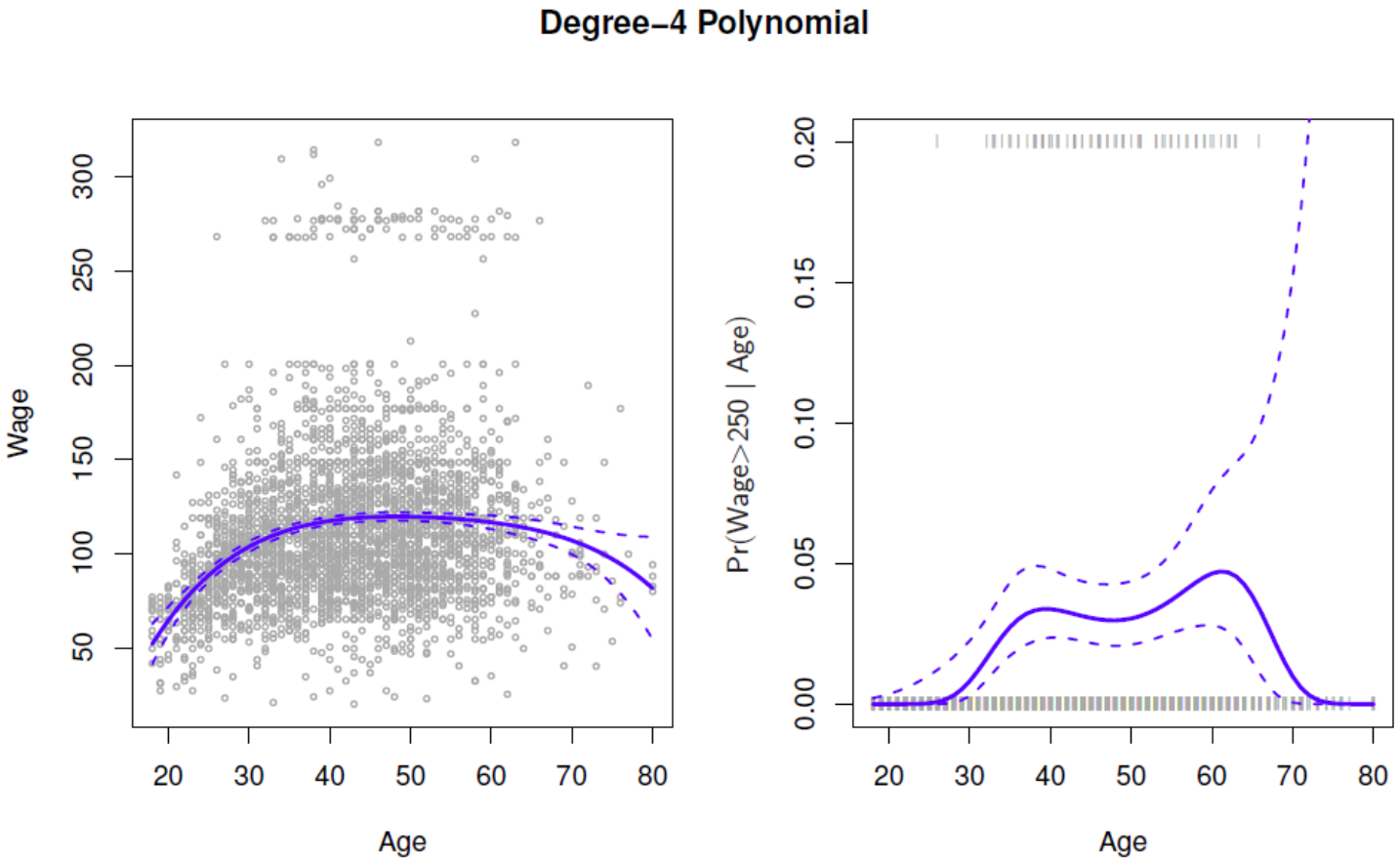
Logistic regression

With more than two classes:

$$\Pr\left(Y = c | \vec{X} = x\right) = \frac{e^{\beta_0^{(c)} + x \cdot \beta^{(c)}}}{\sum_c e^{\beta_0^{(c)} + x \cdot \beta^{(c)}}}$$

Polynomial Regression

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \ldots + \beta_d x_i^d + \epsilon_i$$

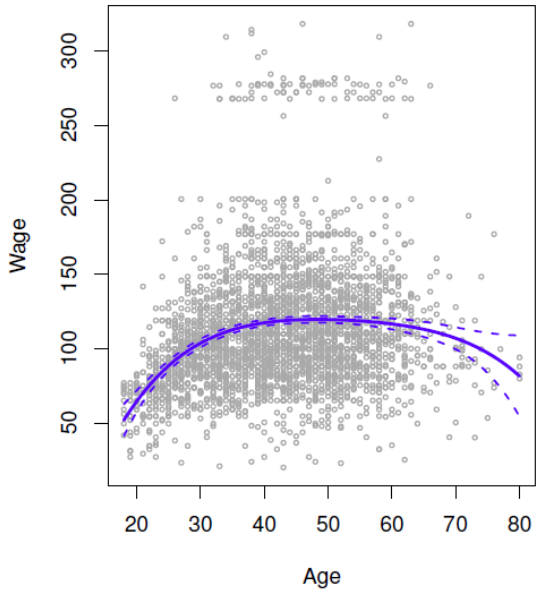


Details

- Create new variables $X_1 = X$, $X_2 = X^2$; etc and then treat as multiple linear regression.
- Not really interested in the coefficients; more interested in the fitted function values at any value x_0:

$$\hat{f}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0 + \hat{\beta}_2 x_0^2 + \hat{\beta}_3 x_0^3 + \hat{\beta}_4 x_0^4.$$

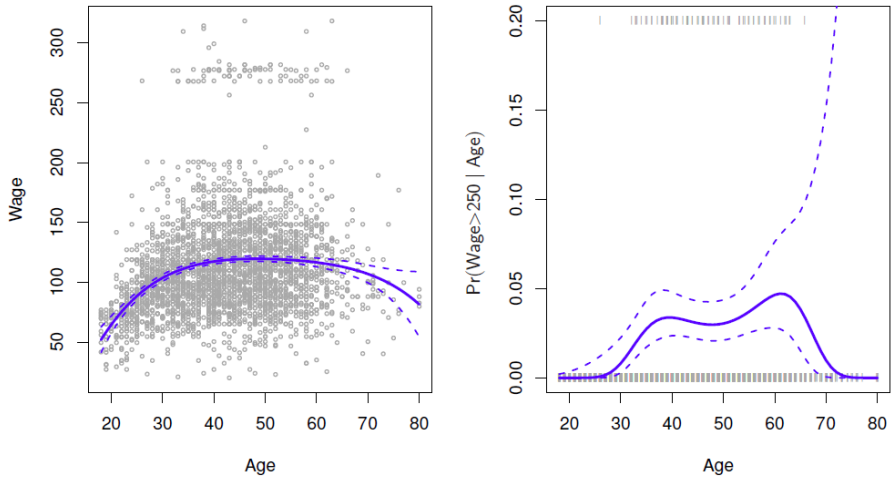
We either fix the degree d at some reasonably low value, else use cross-validation to choose d.



Details continued

- Logistic regression follows naturally. For example, in figure we model

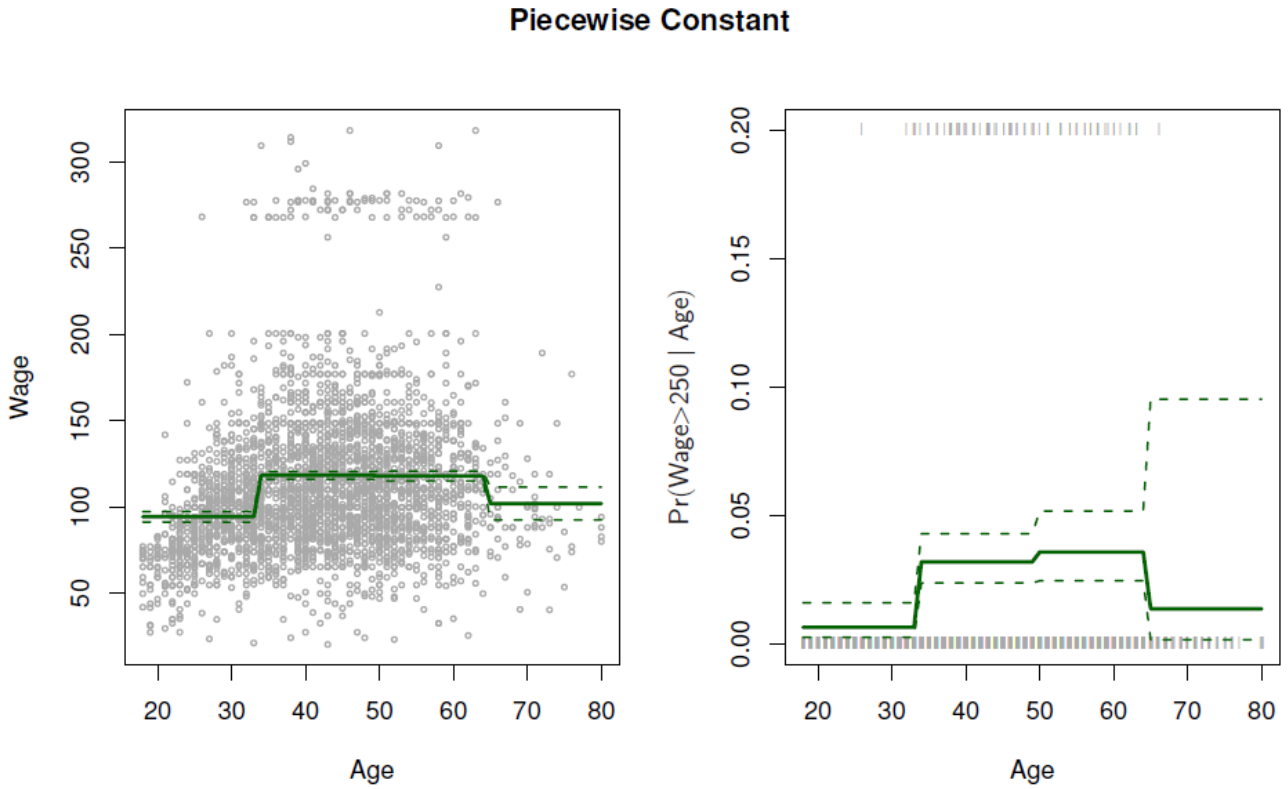
$$\Pr(y_i > 250|x_i) = \frac{\exp(\beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \ldots + \beta_d x_i^d)}{1 + \exp(\beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \ldots + \beta_d x_i^d)}.$$



Step Functions

- Another way of creating transformations of a variable - cut the variable into distinct regions.

$$C_1(X) = I(X < 35), \quad C_2(X) = I(35 \leq X < 50), \ldots, C_3(X) = I(X \geq 65)$$



Step functions continued

- Easy to work with. Creates a series of dummy variables representing each group.

$$y_i = \beta_0 + \beta_1 C_1(x_i) + \beta_2 C_2(x_i) + \ldots + \beta_K C_K(x_i) + \epsilon_i.$$

- Choice of cutpoints or knots can be problematic.

Basis Functions

- Polynomial and piecewise-constant regression models are in fact special cases of a *basis function approach.*

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \beta_3 b_3(x_i) + \ldots + \beta_K b_K(x_i) + \epsilon_i.$$

The basis functions $b_1(\cdot), b_2(\cdot), \ldots, b_K(\cdot)$ *are fixed and known*

Regression Splines

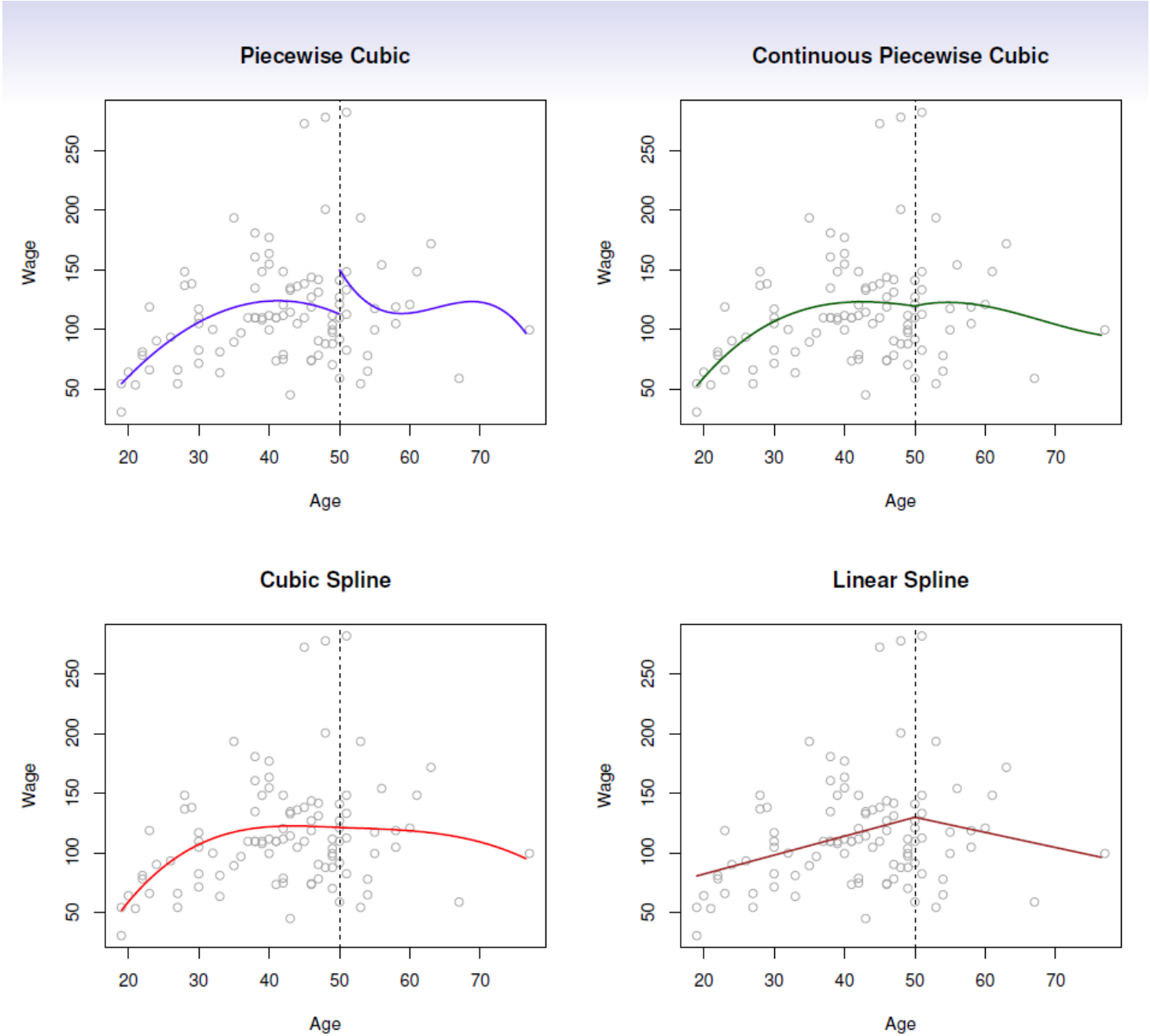
- Now we discuss a flexible class of basis functions that extends upon the polynomial regression and piecewise constant regression approaches that we have just seen.

Piecewise Polynomials

- Instead of a single polynomial in X over its whole domain, we can rather use different polynomials in regions defined by knots

$$y_i = \begin{cases} \beta_{01} + \beta_{11} x_i + \beta_{21} x_i^2 + \beta_{31} x_i^3 + \epsilon_i & \text{if } x_i < c; \\ \beta_{02} + \beta_{12} x_i + \beta_{22} x_i^2 + \beta_{32} x_i^3 + \epsilon_i & \text{if } x_i \geq c. \end{cases}$$

- Using more knots leads to a more flexible piecewise polynomial.
- Better to add constraints to the polynomials, e.g. continuity.
- Splines have the “maximum" amount of continuity.


Piecewise Cubic
Continuous Piecewise Cubic
Cubic Spline
Linear Spline
Wage
Age
50
100
150
200
250
20
30
40
50
60
70

Linear Splines

- A linear spline with knots at ξ_k; k = 1…K is a piecewise linear polynomial continuous at each knot.
- We can represent this model as

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \ldots + \beta_{K+1} b_{K+1}(x_i) + \varepsilon_i,$$

where the b_k are basis functions.

$$\begin{aligned} b_1(x_i) &= x_i \\ b_{k+1}(x_i) &= (x_i - \xi_k)_+, \quad k = 1, \ldots, K \end{aligned}$$

Here the $()_+$ means positive part; i.e.

$$(x_i - \xi_k)_+ = \begin{cases} x_i - \xi_k & \text{if } x_i > \xi_k \\ 0 & \text{otherwise} \end{cases}$$

Cubic Splines

- A cubic spline with knots at ξ_k; k = 1,…,K is a piecewise cubic polynomial with continuous derivatives up to order 2 at each knot.
- Again we can represent this model with truncated power basis functions

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \cdots + \beta_{K+3} b_{K+3}(x_i) + \epsilon_i,$$

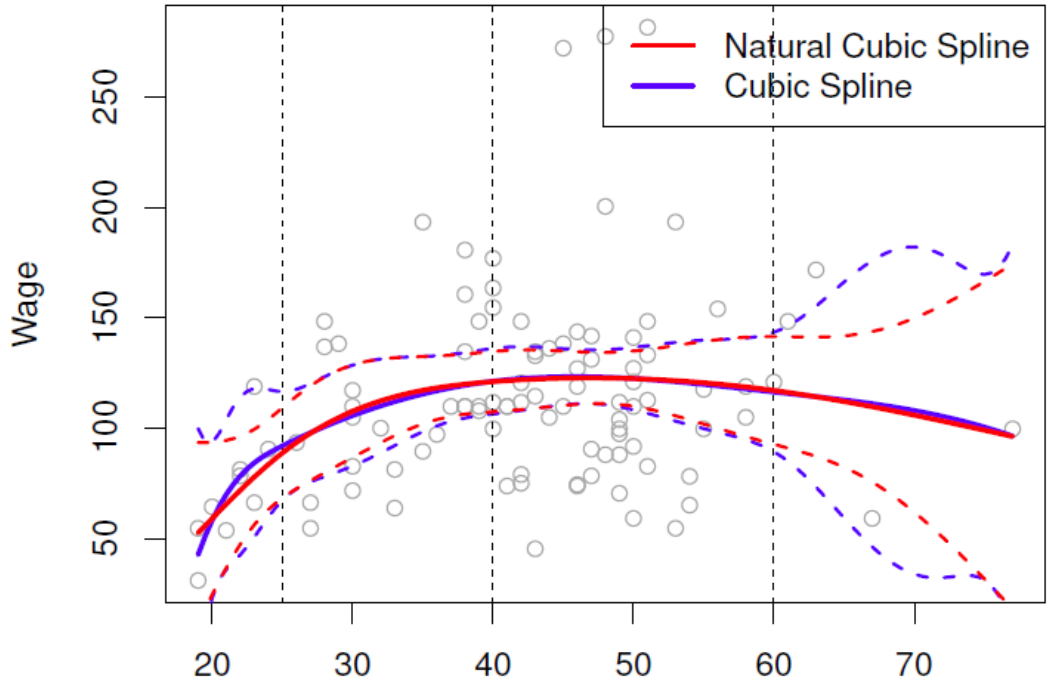
$$\begin{aligned} b_1(x_i) &= x_i \\ b_2(x_i) &= x_i^2 \\ b_3(x_i) &= x_i^3 \\ b_{k+3}(x_i) &= (x_i - \xi_k)_+^3, \quad k = 1, \ldots, K \end{aligned}$$

where

$$(x_i - \xi_k)_+^3 = \begin{cases} (x_i - \xi_k)^3 & \text{if } x_i > \xi_k \\ 0 & \text{otherwise} \end{cases}$$

Natural Cubic Splines

- A natural cubic spline extrapolates linearly beyond the boundary knots. This adds 4 = 2x2 extra constraints, and allows us to put more internal knots for the same degrees of freedom as a regular cubic spline.



Smoothing Splines

- Consider this criterion for fitting a smooth function g(x) to some data:

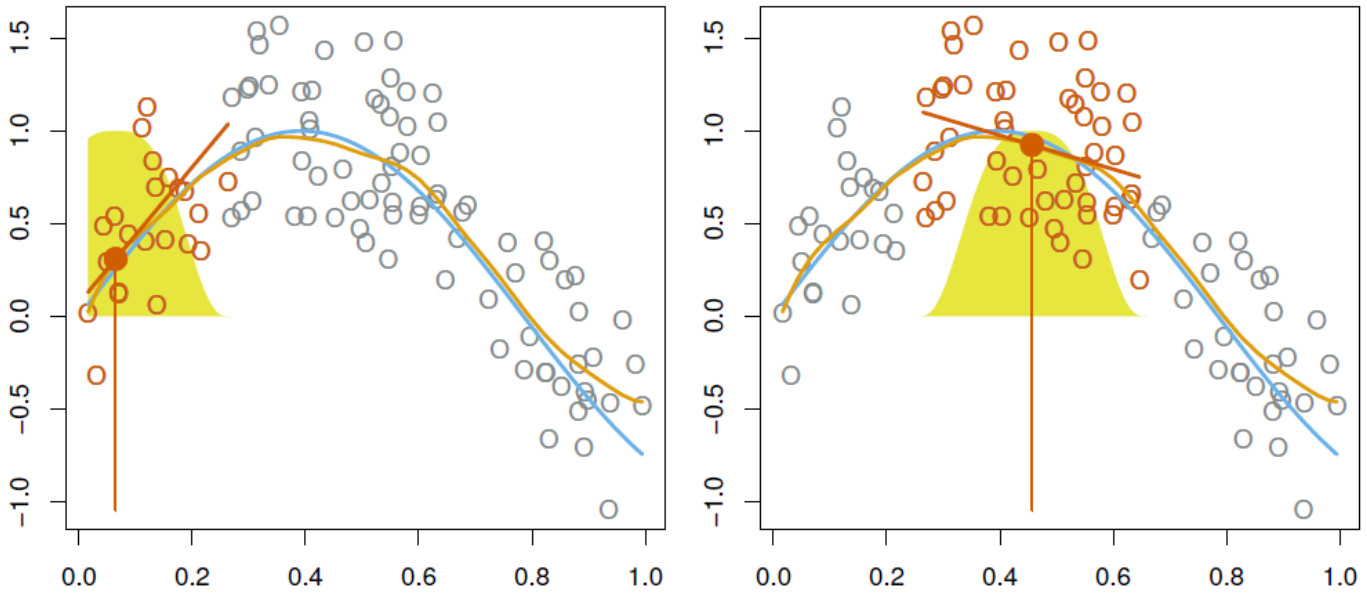
$$\underset{g \in \mathcal{S}}{\text{minimize}} \sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int g''(t)^2 dt$$

- The first term is RSS and tries to make g(x) match the data at each x_i.
- The second term is a roughness penalty and controls how wiggly g(x) is. It is modulated by the tuning parameter $\lambda \geq 0$.
  - The smaller λ, the more wiggly the function, eventually interpolating y_i when = 0.
  - As $\lambda \rightarrow \infty$, the function g(x) becomes linear.

Smoothing Splines continued

- The solution is a natural cubic spline, with a knot at every unique value of x_i. The roughness penalty still controls the roughness via λ.
- Some details
  - Smoothing splines avoid the knot-selection issue, leaving a single λ to be chosen.
  - The algorithmic details are too complex to describe here. In R, the function smooth.spline() will fit a smoothing spline.

Local Regression



- With a sliding weight function, we fit separate linear fits over the range of X by weighted least squares.

Local Regression

1. Gather the fraction $s = k/n$ of training points whose x_i are closest to x_0.

2. Assign a weight $K_{i0} = K(x_i, x_0)$ to each point in this neighborhood, so that the point furthest from x_0 has weight zero, and the closest has the highest weight. All but these k nearest neighbors get weight zero.

3. Fit a *weighted least squares regression* of the y_i on the x_i using the aforementioned weights, by finding $\hat{\beta}_0$ and $\hat{\beta}_1$ that minimize

$$\sum_{i=1}^{n} K_{i0}(y_i - \beta_0 - \beta_1 x_i)^2.$$

4. The fitted value at x_0 is given by $\hat{f}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0$.

Generalized Additive Models

- Allows for flexible nonlinearities in several variables but retains the additive structure of linear models.

- A natural way to extend the multiple linear regression model

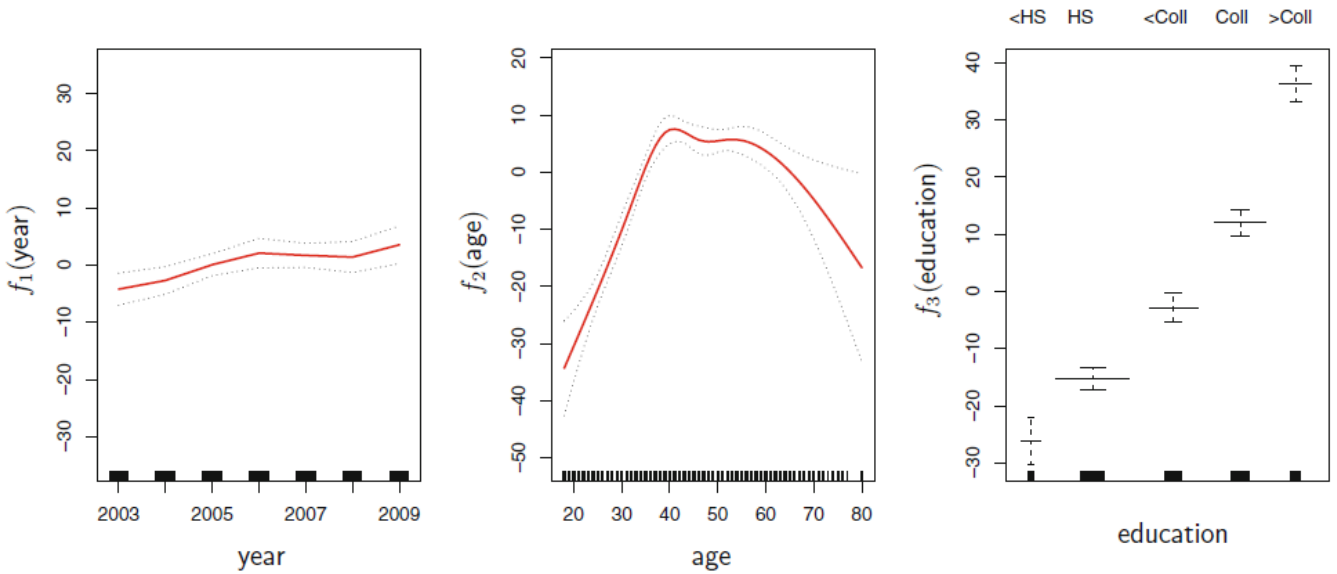
$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \epsilon_i$$

is

$$\begin{aligned} y_i &= \beta_0 + \sum_{j=1}^{p} f_j(x_{ij}) + \epsilon_i \\ &= \beta_0 + f_1(x_{i1}) + f_2(x_{i2}) + \cdots + f_p(x_{ip}) + \epsilon_i. \end{aligned}$$

Generalized Additive Models

$$\texttt{wage} = \beta_0 + f_1(\texttt{year}) + f_2(\texttt{age}) + f_3(\texttt{education}) + \epsilon$$



wage tends to increase slightly with year; this may be due to inflation,
wage tends to be highest for intermediate values of age, and lowest for the very young and very old,
wage tends to increase with education: the more educated a person is, the higher their salary, on average.

Pros and Cons of GAMs

- GAMs allow us to fit a non-linear function *to each variable, so that we can* automatically model non-linear relationships that standard linear regression will miss.
- The non-linear fits can potentially make more accurate predictions for the response *Y.*
- Because the model is additive, we can still examine the effect of each *variable on Y individually while holding all of the other variables* fixed
- The main limitation of GAMs is that the model is restricted to be additive.

GAMs for Classification Problems

- GAMs can also be used in situations where *Y* is qualitative.
- For simplicity, here we will assume *Y* takes on values zero or one, and let $p(X) = P(Y=1|X)$ be the conditional probability (given the predictors) that the response equals one.

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + f_1(X_1) + f_2(X_2) + \cdots + f_p(X_p).$$

GAMs for Classification Problems

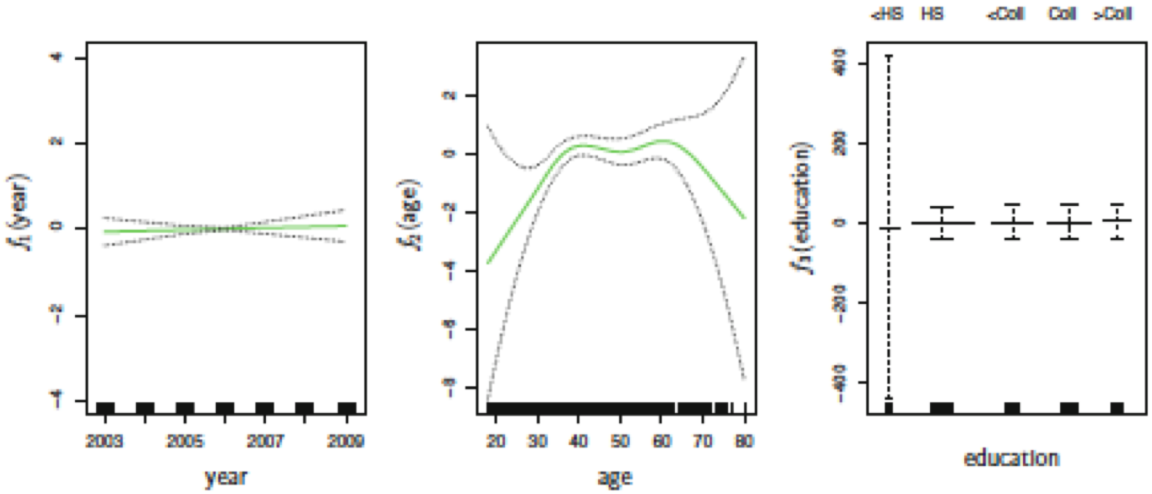
- We fit a GAM to the Wage data in order to predict the probability that an individual's income exceeds $250,000 per year. The GAM that we fit takes the form

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 \times \texttt{year} + f_2(\texttt{age}) + f_3(\texttt{education})$$

Where

$$p(X) = \Pr(\texttt{wage} > 250 | \texttt{year}, \texttt{age}, \texttt{education}).$$

- *The first function is linear in* year*, the second function a smoothing spline with five degrees of freedom in* age*, and the third a step function for* education*.*

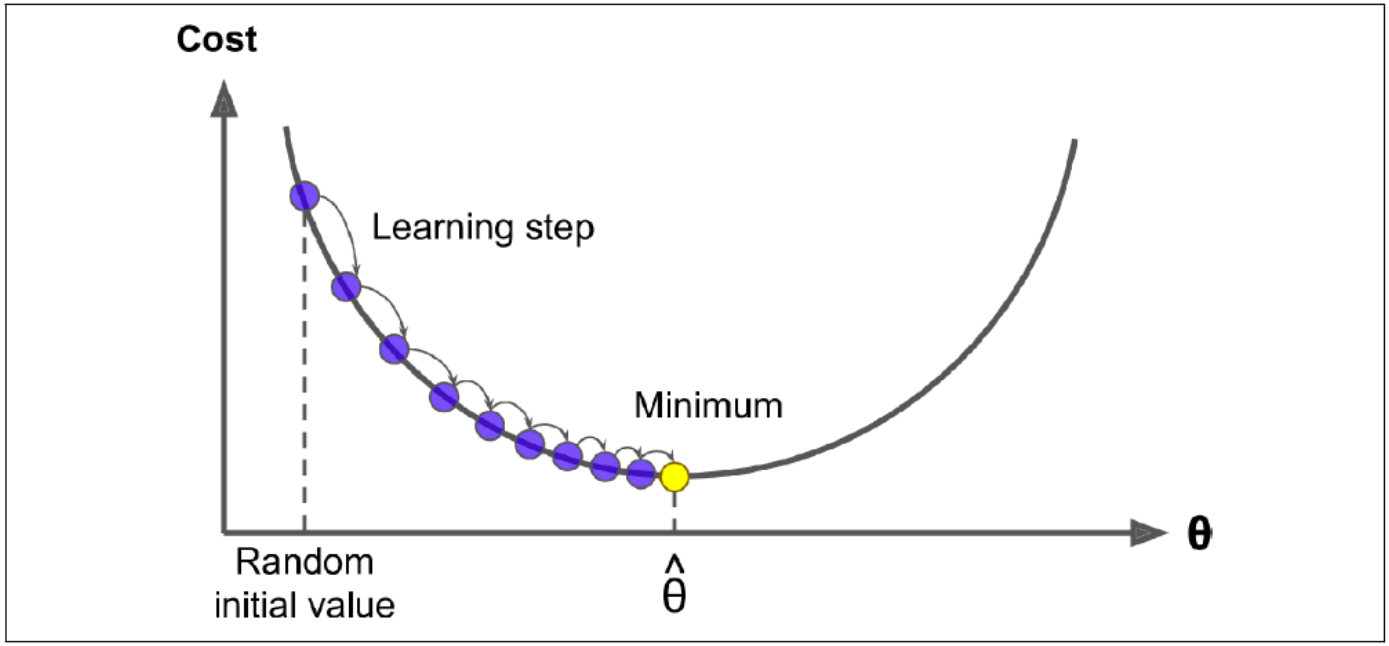


GRADIENT DESCENT

Minería de Datos: Preprocesamiento y clasificación

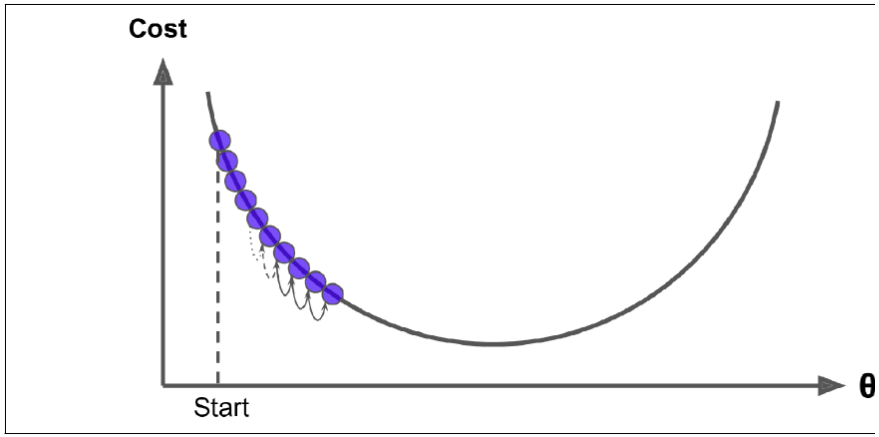
Gradient Descent

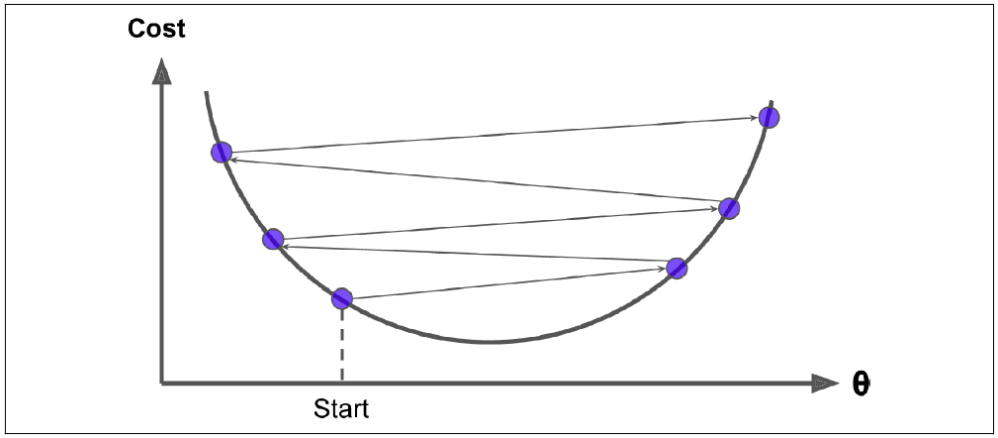
- Gradient Descent is a generic optimization algorithm capable of finding optimal solutions to a wide range of problems. The general idea of Gradient Descent is to tweak parameters iteratively in order to minimize a cost function.
- It measures the local gradient of the error function with regard to the parameter vector θ, and it goes in the direction of descending gradient. Once the gradient is zero, you have reached a minimum!



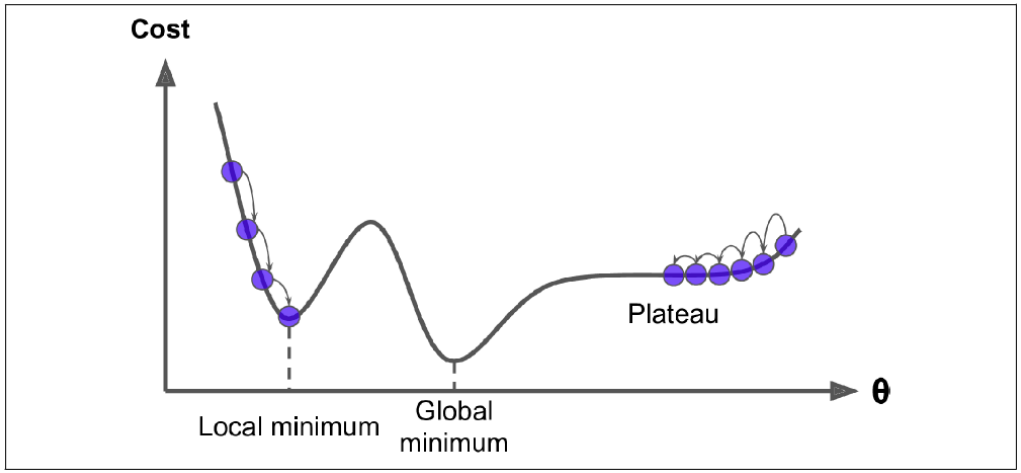
Gradient Descent

- An important parameter in Gradient Descent is the size of the steps, determined by the *learning rate* hyperparameter.



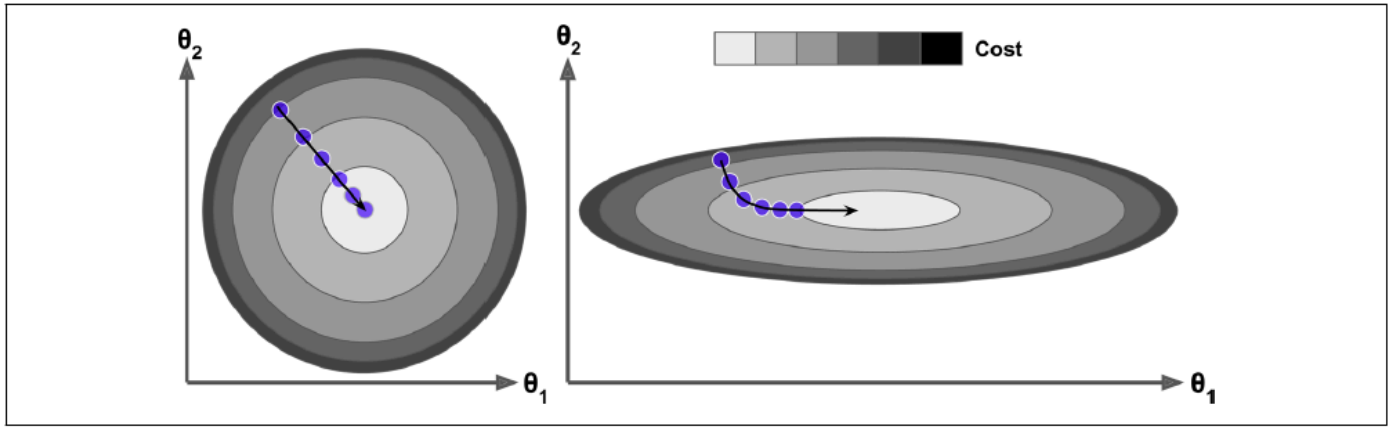


- Not all cost functions look like nice, regular bowls. There may be holes, ridges, plateaus, and all sorts of irregular terrains, making convergence to the minimum difficult



Gradient Descent

- Ej: The MSE cost function for a Linear Regression model happens to be a convex function, which means that if you pick any two points on the curve, the line segment joining them never crosses the curve. This implies that there are no local minima, just one global minimum. It is also a continuous function with a slope that never changes abruptly. These two facts have a great consequence: Gradient Descent is guaranteed to approach arbitrarily close the global minimum (if you wait long enough and if the learning rate is not too high).
- Importance of feature scaling.



Batch Gradient Descent

- Compute the gradient of the cost function with regard to each model parameter θ_j.
- You need to calculate how much the cost function will change if you change θ_j just a little bit. This is called a partial derivative.

$$\frac{\partial}{\partial \theta_j} \text{MSE}(\boldsymbol{\theta}) = \frac{2}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta}^\intercal \mathbf{x}^{(i)} - y^{(i)} \right) x_j^{(i)}$$

- Compute them all in one go. The gradient vector, noted $\nabla_\theta \text{MSE}(\theta)$, contains all the partial derivatives of the cost function

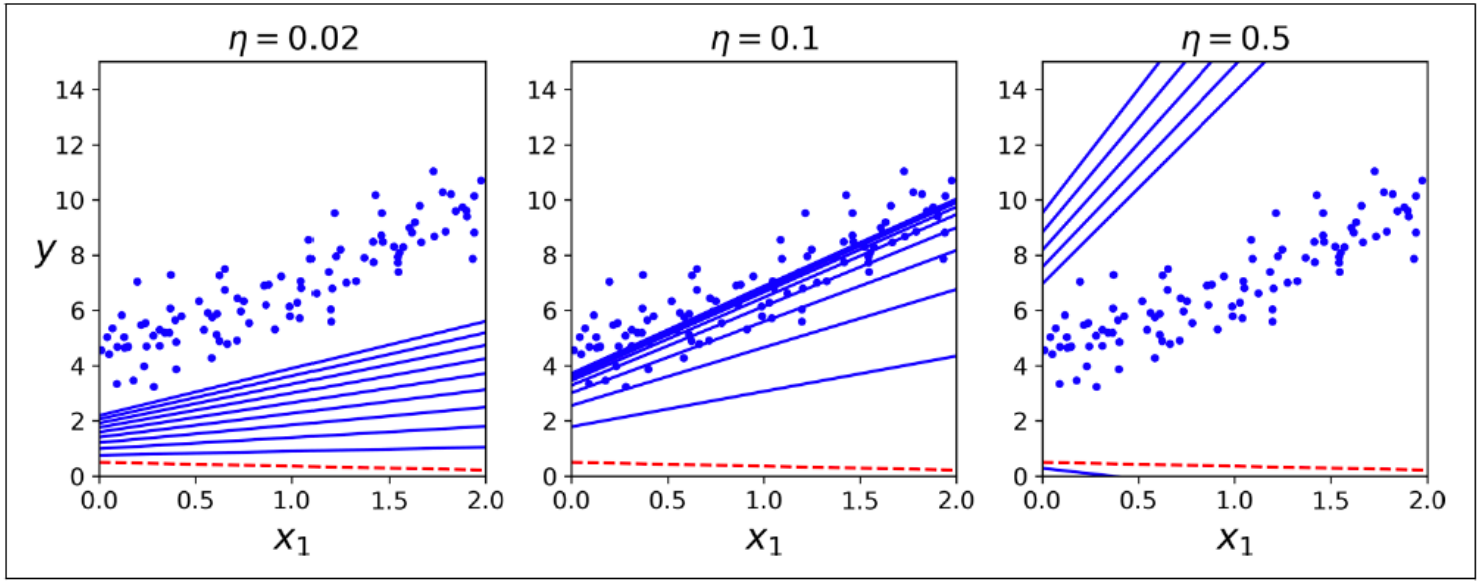
$$\nabla_{\boldsymbol{\theta}} \text{MSE}(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \text{MSE}(\boldsymbol{\theta}) \\ \frac{\partial}{\partial \theta_1} \text{MSE}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \text{MSE}(\boldsymbol{\theta}) \end{pmatrix} = \frac{2}{m} \mathbf{X}^\intercal (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

This formula involves calculations over the full training set X, at each Gradient Descent step! This is why the algorithm is called Batch Gradient Descent.

Batch Gradient Descent

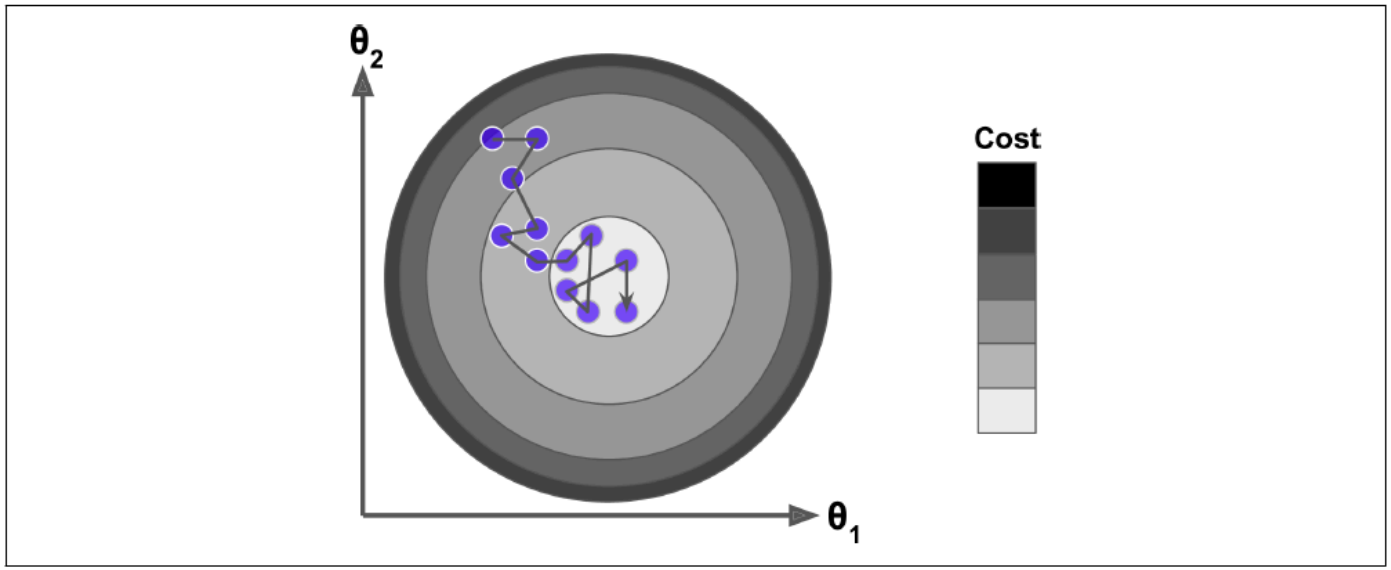
- Once you have the gradient vector, which points uphill, just go in the opposite direction to go downhill. This means subtracting $\nabla_\theta \text{MSE}(\boldsymbol{\theta})$ from $\boldsymbol{\theta}$. This is where the learning rate η comes into play.

$$\boldsymbol{\theta}^{(\text{next step})} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \text{MSE}(\boldsymbol{\theta})$$



Stochastic Batch Gradient Descent

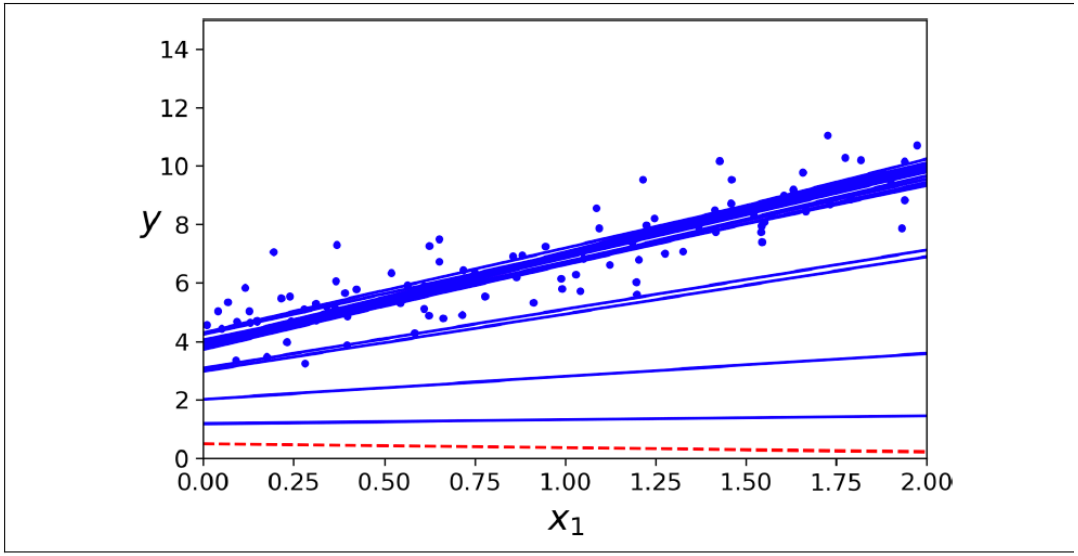
- Picks a random instance in the training set at every step and computes the gradients based only on that single instance.
  - Working on a single instance at a time makes the algorithm much faster because it has very little data to manipulate at every iteration.
  - It also makes it possible to train on huge training sets, since only one instance needs to be in memory at each iteration.
  - This algorithm is much less regular than Batch Gradient Descent: instead of gently decreasing until it reaches the minimum, the cost function will bounce up and down, decreasing only on average.



Stochastic Batch Gradient Descent

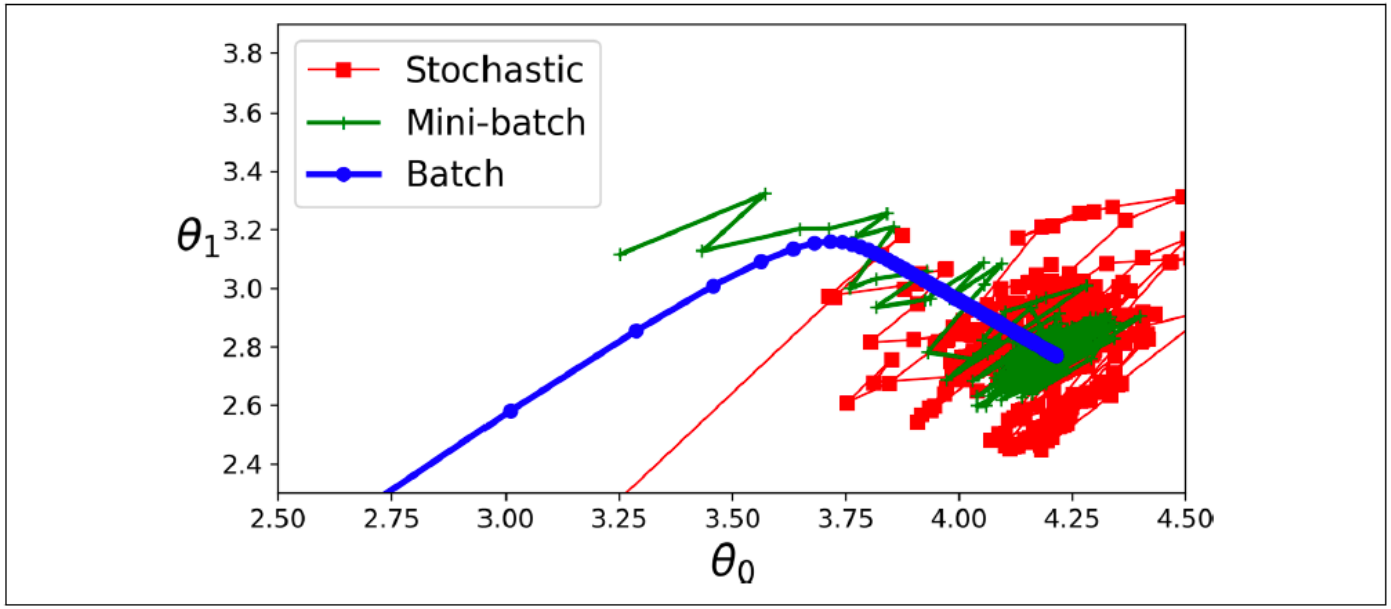
- When the cost function is very irregular, it can actually help the algorithm jump out of local mínima. But the algorithm can never settle at the mínimum.
- Solution: gradually reduce the learning rate. The steps start out large (which helps make quick progress and escape local minima), then get smaller and smaller.
- Function of *learning schedule*

To perform Linear Regression using Stochastic GD with Scikit-Learn, you can use the `SGDRegressor` class, which defaults to optimizing the squared error cost function. It starts with a learning rate of 0.1 (`eta0=0.1`), using the default learning schedule. Lastly, it does not use any regularization (`penalty=None`)



Mini-batch Gradient Descent

- At each step, instead of computing the gradients based on the full training set (as in Batch GD) or based on just one instance (as in Stochastic GD), Mini-batch GD computes the gradients on small random sets of instances called mini-batches.
- The main advantage of Mini-batch GD over Stochastic GD is that you can get a performance boost from hardware optimization of matrix operations, especially when using GPUs.



REGULARIZED LINEAR MODELS

Minería de Datos: Preprocesamiento y clasificación

Ridge Regression

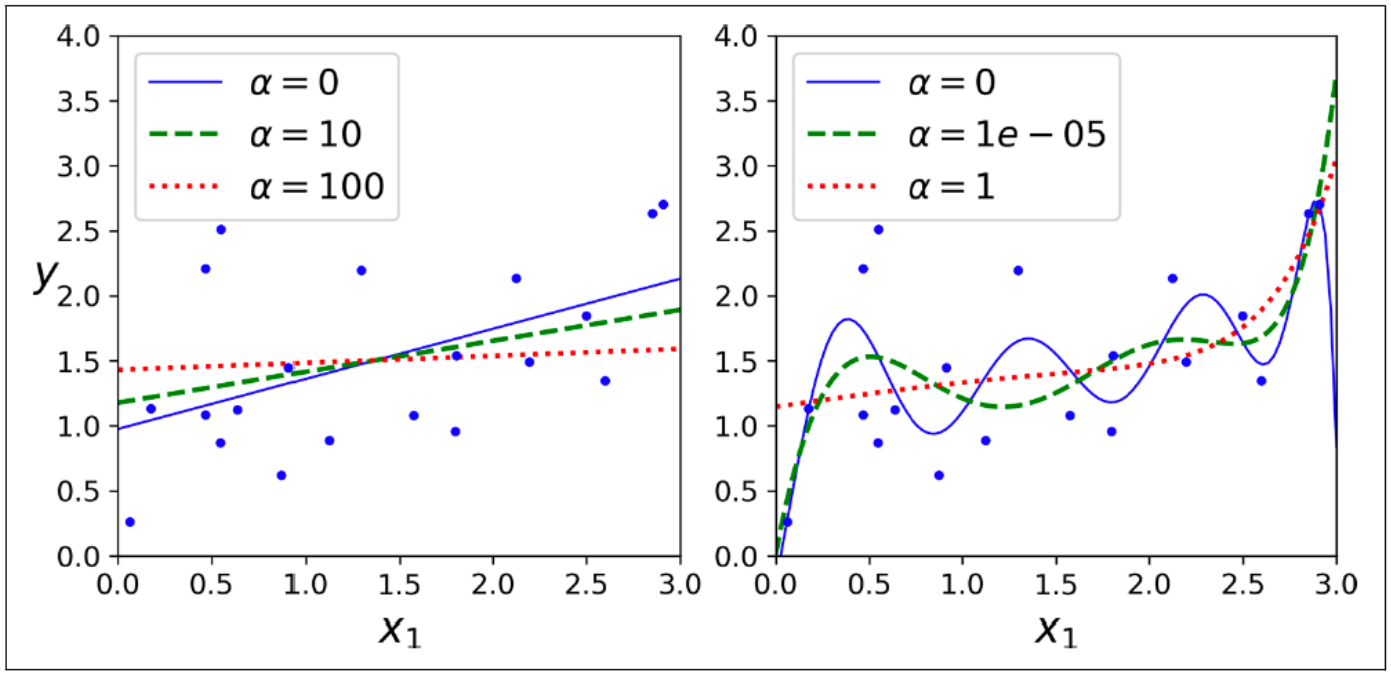
- Regularized version of Linear Regression: a regularization term equal to $\alpha \sum_{i=1}^{n} \theta_i^2$ is added to the cost function.

- This forces the learning algorithm to not only fit the data but also keep the model weights as small as possible.
- The regularization term should only be added to the cost function during training.
- The hyperparameter α controls how much you want to regularize the model. If α = 0, then Ridge Regression is just Linear Regression. If α is very large, then all weights end up very close to zero and the result is a flat line going through the data's mean.

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$$

Ridge Regression

- Note that the bias term θ_0 is not regularized (the sum starts at i = 1, not 0). If we define **w** as the vector of feature weights (θ_1 to θ_n), then the regularization term is equal to $(1/2)(\|w\|_2)^2$, where $\|w\|_2$ represents the ℓ_2 norm of the weight vector.
- For Gradient Descent, just add $\alpha \cdot w$ to the MSE gradient vector.



Ridge Regression

- Ridge Regression closed-form solution:

$$\hat{\theta} = (\mathbf{X}^\intercal \mathbf{X} + \alpha \mathbf{A})^{-1} \quad \mathbf{X}^\intercal \quad \mathbf{y}$$

```
>>> from sklearn.linear_model import Ridge
>>> ridge_reg = Ridge(alpha=1, solver="cholesky")
>>> ridge_reg.fit(X, y)
```

- Using SGD:

```
>>> sgd_reg = SGDRegressor(penalty="l2")
>>> sgd_reg.fit(X, y.ravel())
```

A is the (n + 1) x (n + 1) identity matrix, except with a 0 in the top-left cell, corresponding to the bias term

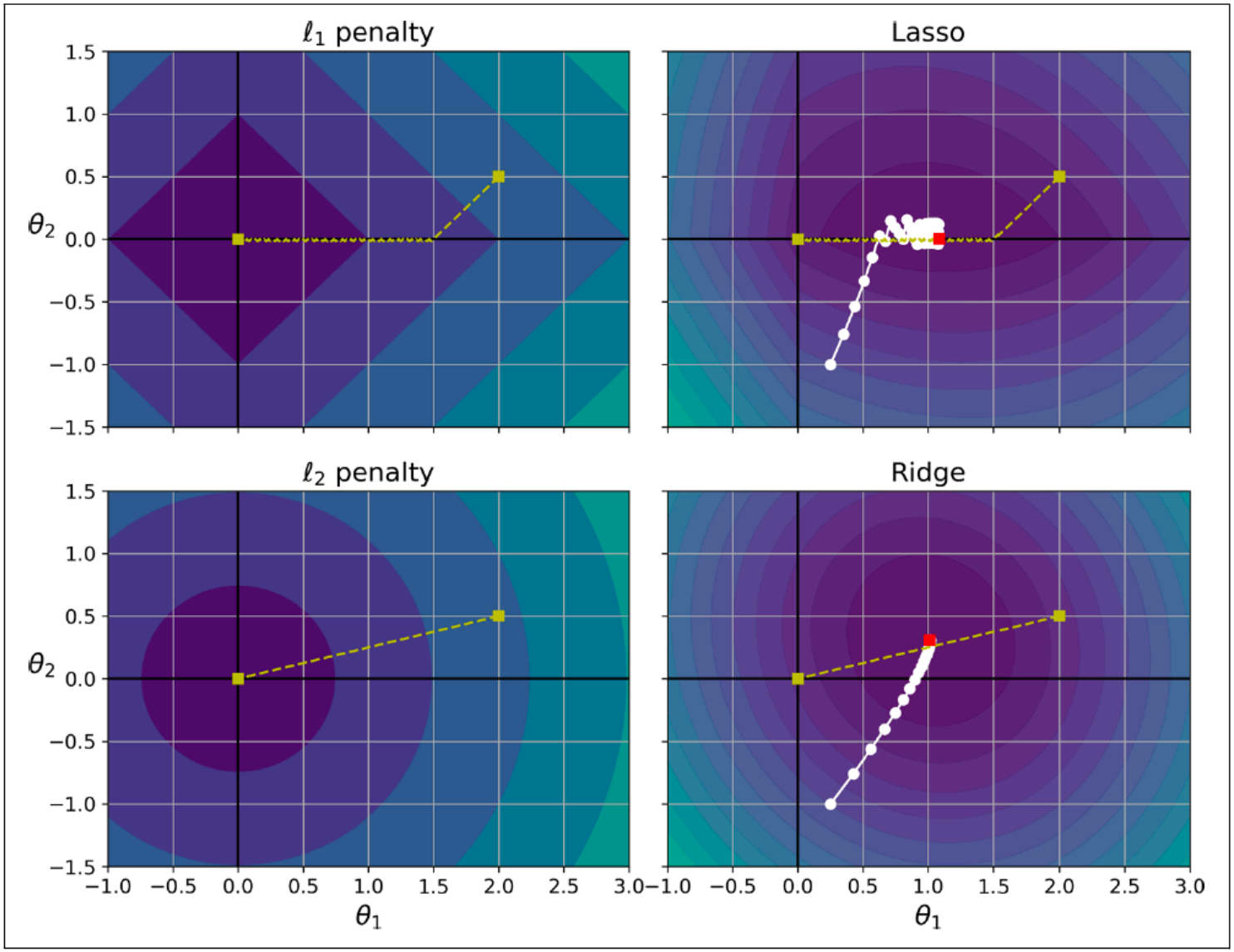
Lasso Regression

- Least Absolute Shrinkage and Selection Operator Regression is another regularized version of Linear Regression.
- Just like Ridge Regression, it adds a regularization term to the cost function, but it uses the ℓ1 norm of the weight vector instead of half the square of the ℓ2 norm.

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + \alpha \sum_{i=1}^{n} \left|\theta_i\right|$$

- An important characteristic of Lasso Regression is that it tends to eliminate the weights of the least important features (i.e., set them to zero).
- In other words, Lasso Regression automatically performs feature selection and outputs a sparse model (i.e., with few nonzero feature weights).

Lasso Regression



Lasso Regression

- The Lasso cost function is not differentiable at $\theta_i = 0$ (for i = 1, 2, ⋯, n), but Gradient Descent still works fine if you use a subgradient vector **g** instead when any $\theta_i = 0$.

$$g(\boldsymbol{\theta}, J) = \nabla_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta}) + \alpha \begin{pmatrix} \operatorname{sign}(\theta_1) \\ \operatorname{sign}(\theta_2) \\ \vdots \\ \operatorname{sign}(\theta_n) \end{pmatrix} \quad \text{where} \quad \operatorname{sign}(\theta_i) = \begin{cases} -1 & \text{if } \theta_i < 0 \\ 0 & \text{if } \theta_i = 0 \\ +1 & \text{if } \theta_i > 0 \end{cases}$$

```
>>> from sklearn.linear_model import Lasso
>>> lasso_reg = Lasso(alpha=0.1)
>>> lasso_reg.fit(X, y)
```

Note that you could instead use `SGDRegressor(penalty="l1")`.

Elastic Net

- Elastic Net is a middle ground between Ridge Regression and Lasso Regression. The regularization term is a simple mix of both Ridge and Lasso's regularization terms, and you can control the mix ratio r. When r = 0, Elastic Net is equivalent to Ridge Regression, and when r = 1, it is equivalent to Lasso Regression.

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + r\alpha\Sigma_{i=1}^{n}\left|\theta_i\right| + \frac{1-r}{2}\alpha\Sigma_{i=1}^{n}\theta_i^2$$

It is almost always preferable to have at least a little bit of regularization, so generally you should avoid plain Linear Regression. Ridge is a Good default, but if you suspect that only a few features are useful, you should prefer Lasso or Elastic Net because they tend to reduce the useless features' weights down to zero, as we have discussed. In general, Elastic Net is preferred over Lasso because Lasso may behave erratically when the number of features is greater than the number of training instances or when several features are strongly correlated.

```
>>> from sklearn.linear_model import ElasticNet
>>> elastic_net = ElasticNet(alpha=0.1, l1_ratio=0.5)
>>> elastic_net.fit(X, y)
```

NEURAL NETWORKS BASICS

Minería de Datos: Preprocesamiento y clasificación

Introducción

El cerebro humano

- Formado por una red de neuronas interconectadas.
- 10^{10} neuronas.
- 10^{4} conexiones por neurona.
- Tiempo de conmutación 0.001 seg.
- Tiempo en reconocer una imagen: 0.1 seg.
- Para resolver el problema de reconocimiento de imágenes, se necesita bastante más de 100 pasos de procesamiento.

➔ El cerebro humano presenta un paralelismo masivo sobre una representación distribuida.

Introducción

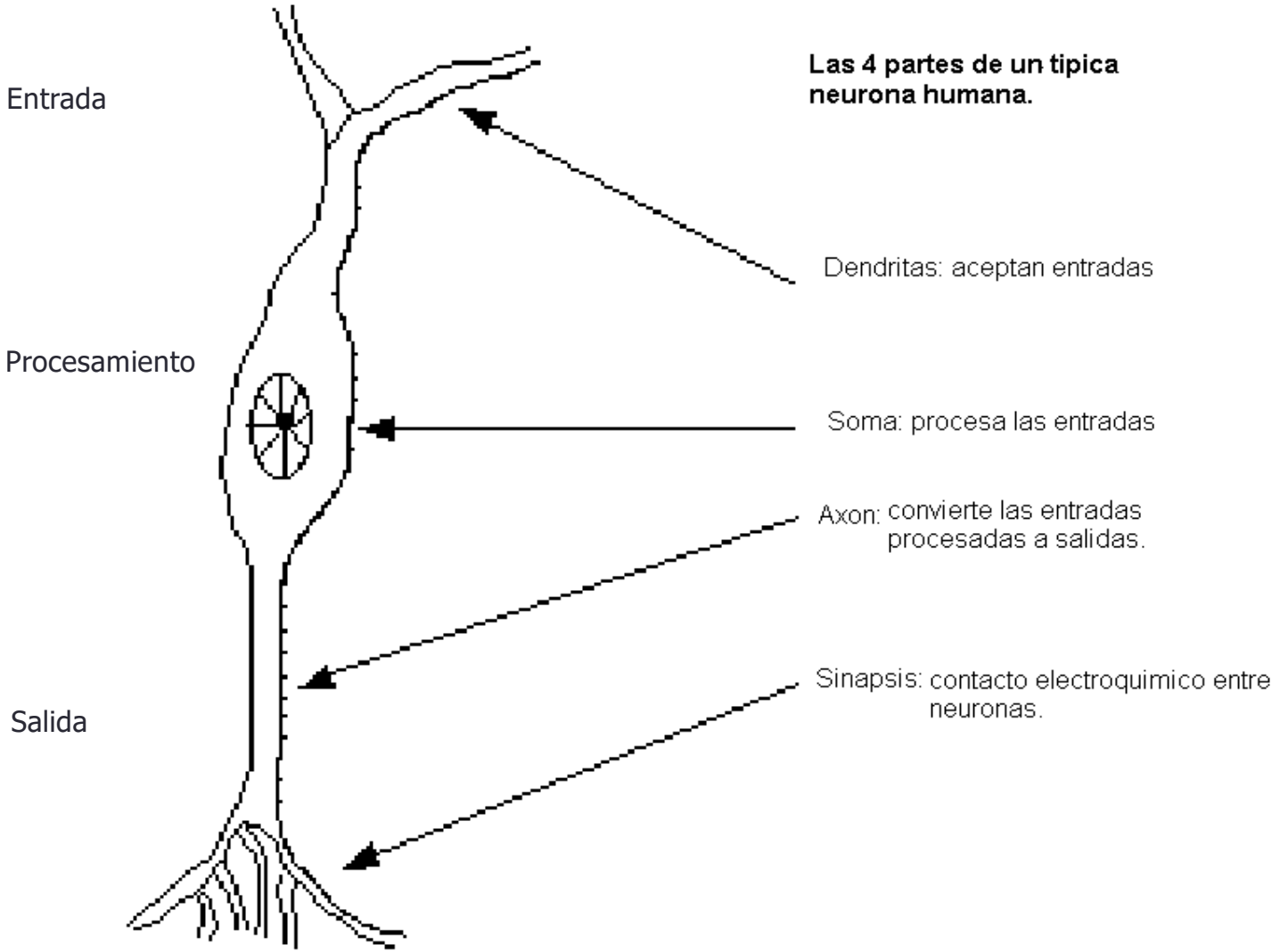
- Excepto en las tareas basadas en el cálculo aritmético simple, actualmente, el cerebro humano es superior a cualquier computador:
  - Procesamiento de imágenes, voz, datos inexactos o con ruido, lenguaje natural, (en general en tareas de percepción)
  - Predicción,
  - Control...
- Características del cerebro:
  - Robusto. Su funcionamiento no se altera ante fallos poco importantes.
  - Flexible. Se adapta con facilidad a un entorno cambiante.
  - Puede tratar con información ambigua o incompleta.
  - Pequeño, compacto y consume poca potencia.

Introducción

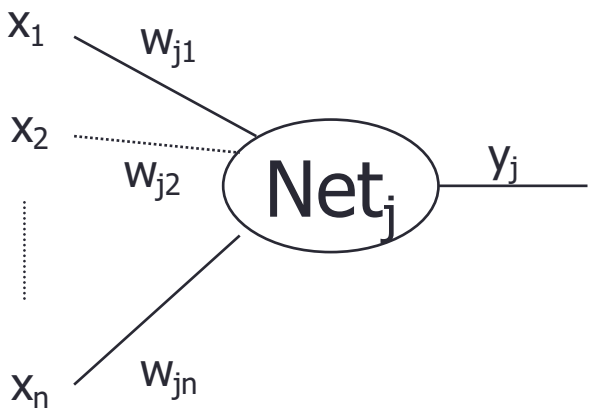
El funcionamiento del cerebro es especialmente significativo en tareas de percepción:

- ¿Cómo podemos reconocer los caracteres escritos, su forma, sin haberlos visto antes?
- ¿o el rostro de alguien desde un ángulo desde el que nunca lo habíamos visto?
- ¿Cómo podemos recordar escenas que hemos visualizado brevemente o de forma vaga?

Las neuronas biológicas



Las neuronas artificiales



- Las señales que llegan a las dendritas se representan como x_1, x_2, ..., x_n.
- Las conexiones sinápticas se representan por unos pesos w_{j1}, w_{j2}, w_{jn} que ponderan (multiplican) a las entradas. Si el peso entre las neuronas j e i es:
  a) positivo, representa una sinapsis excitadora
  b) negativo, representa una sinapsis inhibidora
  c) cero, no hay conexión

- La acción integradora del cuerpo celular (o actividad interna de cada célula) se presenta por

$$Net_j = w_{j1} \cdot x_1 + w_{j2} \cdot x_2 + \ldots + w_{jn} \cdot x_n = \sum_{i=1}^{n} w_{ji} \cdot x_i$$

- La salida de la neurona se representa por y_j. Se obtiene mediante una función que, en general, se denomina función de salida, de transferencia o de activación. Esta función depende de Net_j y de un parámetro θ_j que representa el umbral de activación de la neurona

$$y_j = f(Net_j - \theta_j) = f\left(\sum w_{ji} \cdot x_i - \theta\right)$$

Tipos de funciones de transferencia

- **Función de escalón** o Haviside. Representa una neurona con sólo dos estados de activación: activada (1) y inhibida (0 ó -1)

$$y_j = H(Net_j - \theta_j) = \begin{cases} 1, & si\ Net_j >= \theta_j \\ -1, & si\ Net_j < \theta_j \end{cases}$$

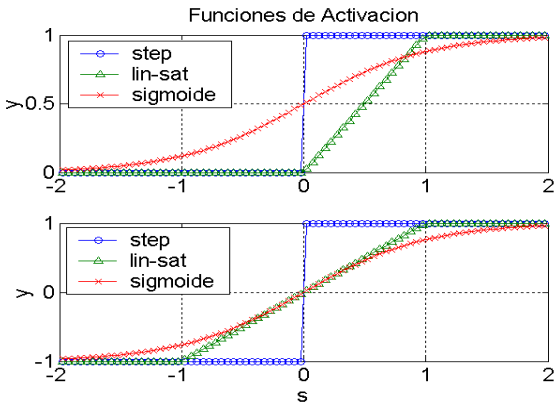
- **Función lineal (ReLU)**: $y_j = Net_j - \theta_j$

- **Función lineal a tramos**:

- **Función sigmoidal (softmax)**:

$$y_j = \begin{cases} 1, & si\ Net_j >= \theta_j + a \\ Net_j - \theta_j, & si\ |Net_j - \theta_j| < a \\ -1, & si\ Net_j < \theta_j - a \end{cases}$$

$$y_j = \frac{1}{1+e^{-\lambda(Net_j-\theta_j)}} \qquad y_j = \frac{2}{1+e^{-\lambda(Net_j-\theta_j)}} - 1$$



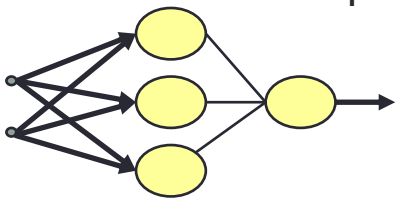
- **Función base radial**:

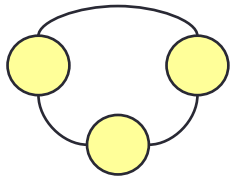
$$y_j = e^{-\left(\frac{Net_j-\theta_j}{\sigma}\right)^2}$$

¿Qué es una red neuronal?

Una **red de neuronas artificiales** está caracterizada por su:

- **Arquitectura**: Estructura o patrón de conexiones entre las unidades de proceso





- **Dinámica de la Computación** que nos expresa el valor que toman las unidades de proceso y que se basa en unas **funciones de activación (o de transferencia)** que especifican como se transforman las señales de entrada de la unidad de proceso en la señal de salida

- **Algoritmo de Entrenamiento o Aprendizaje**: Procedimiento para determinar los pesos de las conexiones.Una característica muy importante de estas redes es su naturaleza ***adaptativa***, donde el **"aprendizaje con ejemplos"** sustituye a la "programación" en la resolución de problemas.

¿Qué es una red neuronal?

Las RNAs poseen las siguientes propiedades:

- No linealidad.
- Permiten la representación de aplicaciones o correspondencias entre las entradas y salidas.
- Adaptabilidad: Acomodan (adaptan) sus pesos sinápticos a los cambios del entorno.
- Información contextual: El conocimiento viene representado por el estado de activación de la red neuronal. Cada neurona está afectada potencialmente por la actividad global de las otras neuronas de la red.

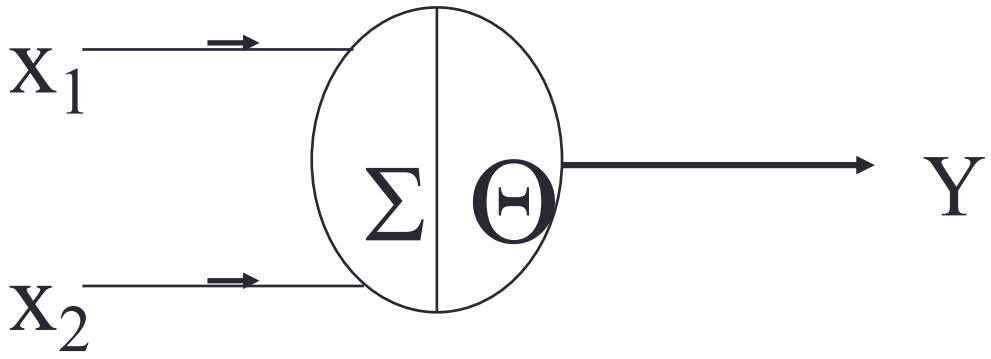
¿Qué es una red neuronal?

- Tolerancia a fallos: Por una parte, si una neurona o un enlace de la red neuronal son dañados, la respuesta de la red probablemente no quedará afectada.
  - Una persona nace con unos 100 billones de neuronas y a partir de los cuarenta años, o antes, se pierden más de 40.000 neuronas diarias, que no son reemplazadas y, sin embargo, continuamos aprendiendo.
  - Una red es capaz de reconocer señales de entrada diferentes a las señales entrenadas cuando difieren moderadamente. Análogamente, una persona puede reconocer a otra después de muchos años sin verla.

- Implementación VLSI: Al ser las redes masivamente paralelas pueden conseguir rapidez de cómputo en la realización de ciertas tareas. Permiten la implementación usando la tecnología VLSI (Very-Large-Scale-Integrated) y conseguir aplicaciones, como el reconocimiento de patrones, procesamiento de señales y control en tiempo real.

Ejemplo de funcionamiento de una neurona

- Ejemplo: neurona simple con dos entradas y una salida. Perceptrón.



Ejemplo de funcionamiento de una neurona

- Intentemos implementar la función OR con la ayuda de esta neurona

X1	X2	Salida
0	0	0
0	1	1
1	0	1
1	1	1

Ejemplo de funcionamiento de una neurona

- Función de transferencia: Función escalón.
- Umbral: 0.5.
- Inicialmente los pesos tienen un valor establecido.
- Los pesos se actualizan con esta ecuación

$$W_i = W_i + \alpha D x_i \ , i = 1,2$$

donde α = parametro de entrenamiento y D es la diferencia entre la salida deseada y la salida real de la neurona.

- El objetivo es encontrar los valores de w_1 y w_2, tal que la salida sea la correcta.

Ejemplo de funcionamiento de una neurona

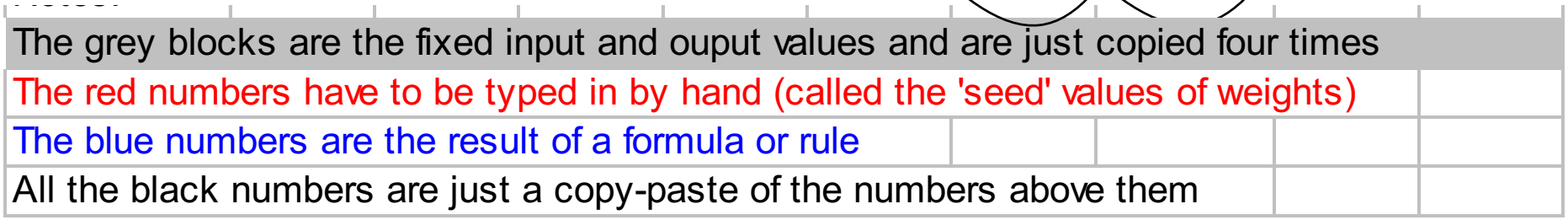
Example of Supervised Learning

(A single neuron that learns the inclusive OR operation)

Parameters: a = 0.2 (measure of the learning rate)

Threshold = 0.5, D = Z - Y, Y = W1 X1 + W2 X2 Wi(Final) = Wi(Initial) + a * D * Xi

			Desired	Initial		Actual	Difference	Final	
	Inputs		Output	Weights		Output		Weights	
Iteration	X1	X2	Z	W1	W2	Y	D	W1	W2
1	0	0	0	0.1	0.3	0	0	0.1	0.3
	0	1	1	0.1	0.3	0	1	0.1	0.5
	1	0	1	0.1	0.5	0	1	0.3	0.5
	1	1	1	0.3	0.5	1	0	0.3	0.5



The grey blocks are the fixed input and ouput values and are just copied four times

The red numbers have to be typed in by hand (called the 'seed' values of weights)

The blue numbers are the result of a formula or rule

All the black numbers are just a copy-paste of the numbers above them

Ejemplo de funcionamiento de una neurona

Example of Supervised Learning

(A single neuron that learns the inclusive OR operation)

Parameters: a = 0.2 (measure of the learning rate)

Threshold = 0.5, D = Z - Y, Y = W1 X1 + W2 X2 Wi(Final) = Wi(Initial) + a * D * Xi

	Inputs		Desired Output	Initial Weights		Actual Output	Difference	Final Weights	
Iteration	X1	X2	Z	W1	W2	Y	D	W1	W2
1	0	0	0	0.1	0.3	0	0	0.1	0.3
	0	1	1	0.1	0.3	0	1	0.1	0.5
	1	0	1	0.1	0.5	0	1	0.3	0.5
	1	1	1	0.3	0.5	1	0	0.3	0.5
2	0	0	0	0.3	0.5	0	0	0.3	0.5
	0	1	1	0.3	0.5	0	1	0.3	0.7
	1	0	1	0.3	0.7	0	1	0.5	0.7
	1	1	1	0.5	0.7	1	0	0.5	0.7
3	0	0	0	0.5	0.7	0	0	0.5	0.7
	0	1	1	0.5	0.7	1	0	0.5	0.7
	1	0	1	0.5	0.7	0	1	0.7	0.7
	1	1	1	0.7	0.7	1	0	0.7	0.7
4	0	0	0	0.7	0.7	0	0	0.7	0.7
	0	1	1	0.7	0.7	1	0	0.7	0.7
	1	0	1	0.7	0.7	1	0	0.7	0.7
	1	1	1	0.7	0.7	1	0	0.7	0.7

Ejemplo de funcionamiento de una neurona

- Encontramos que la ecuación correcta es

$$a = 0.7\ x_1 + 0.7\ x_2$$

- Técnicas como esta pueden se pueden aplicar para resolver un numero extremadamente grande de problemas.

Ejemplo de funcionamiento de una neurona

Ejemplo 2: Función NAND

X1	X2	Z
0	0	1
0	1	1
1	0	1
1	1	0

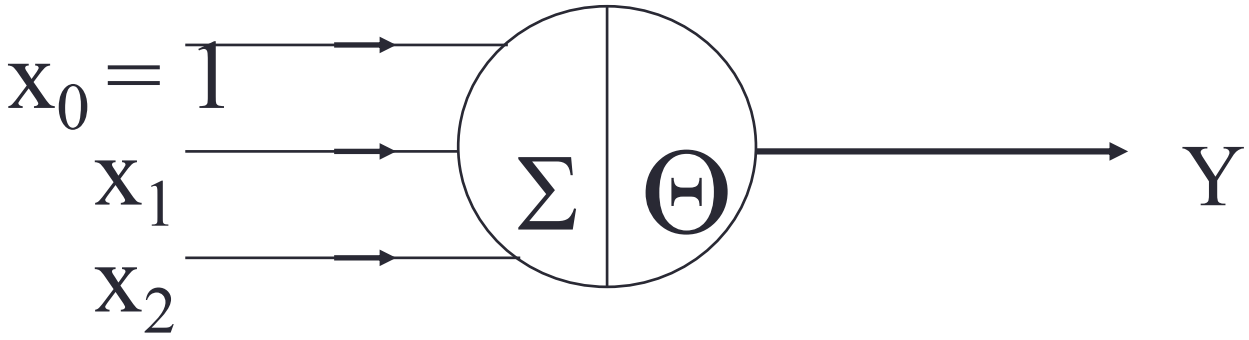
☹ ¿Cómo podemos obtener un 1 de la ecuación

$$Y = w1\ x1 + w2\ x2$$

cuando x1 y x2 son cero?

Ejemplo de funcionamiento de una neurona

Solución: Introducir un término de sesgo



Ejemplo de funcionamiento de una neurona

Example of Supervised Learning

(A single neuron that learns the NAND operation)

Parameters: a = 0.1 (measure of the learning rate)

Threshold = 0.5

D = Z-Y

Y = W0 + W1 X1 + W2 X2 (X0 = 1)

Wi(Final) = Wi(Initial) + a * D * Xi (i = 0,1,2)

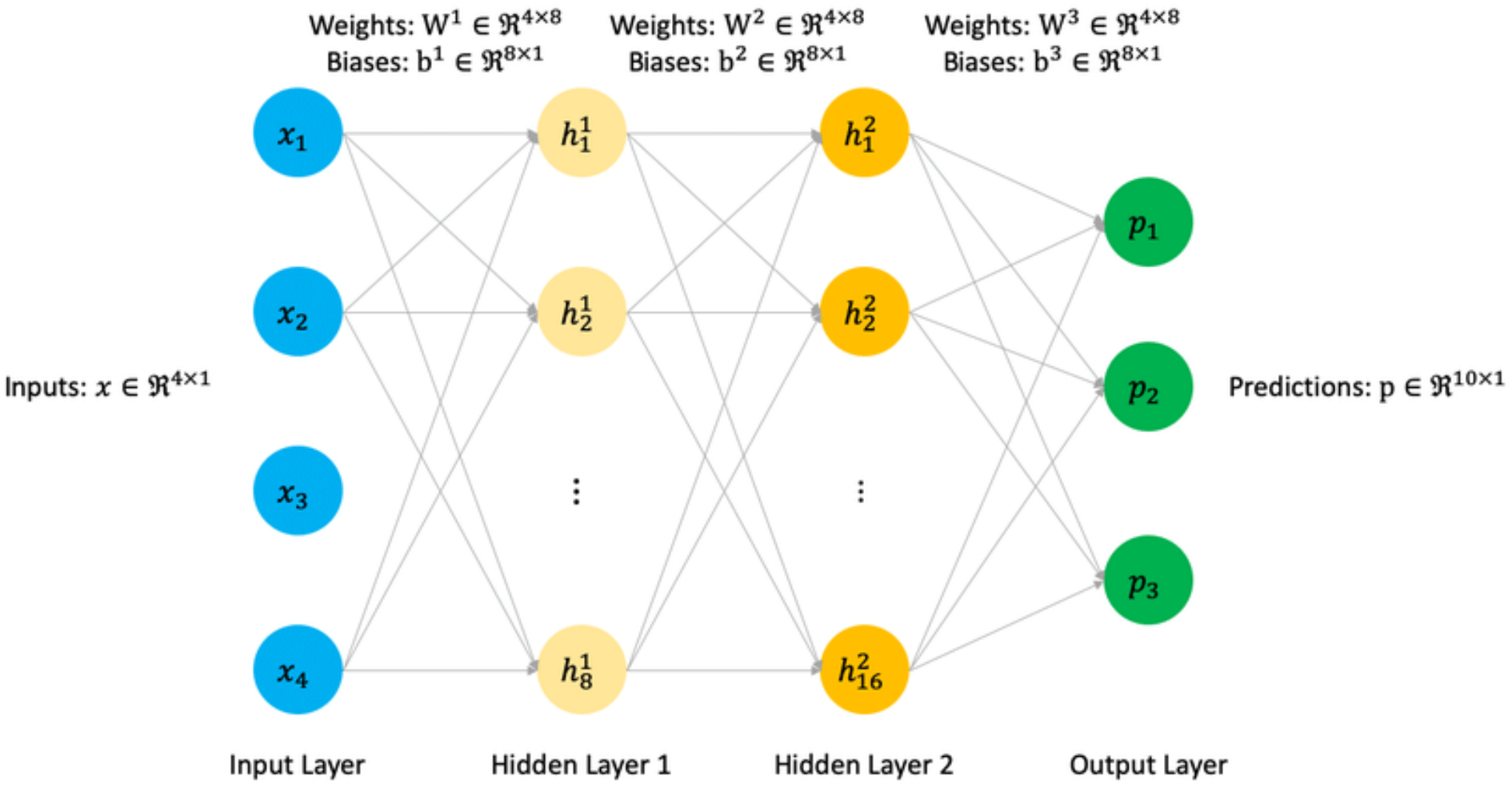
				Desired	Initial			Actual	Difference	Final		
	Inputs			Output	Weights			Output		Weights		
Iteration	X0	X1	X2	Z	W0	W1	W2	Y	D	W0	W1	W2
1	1	0	0	1	0.9	0.1	0.1	1	0	0.9	0.1	0.1
	1	0	1	1	0.9	0.1	0.1	1	0	0.9	0.1	0.1
	1	1	0	1	0.9	0.1	0.1	1	0	0.9	0.1	0.1
	1	1	1	0	0.9	0.1	0.1	1	-1	0.8	0	0
2	1	0	0	1	0.8	0	0	1	0	0.8	0	0
	1	0	1	1	0.8	0	0	1	0	0.8	0	0
	1	1	0	1	0.8	0	0	1	0	0.8	0	0
	1	1	1	0	0.8	0	0	1	-1	0.7	-0.1	-0.1
3	1	0	0	1	0.7	-0.1	-0.1	1	0	0.7	-0.1	-0.1
	1	0	1	1	0.7	-0.1	-0.1	1	0	0.7	-0.1	-0.1
	1	1	0	1	0.7	-0.1	-0.1	1	0	0.7	-0.1	-0.1
	1	1	1	0	0.7	-0.1	-0.1	0	0	0.7	-0.1	-0.1
4	1	0	0	1	0.7	-0.1	-0.1	1	0	0.7	-0.1	-0.1
	1	0	1	1	0.7	-0.1	-0.1	1	0	0.7	-0.1	-0.1
	1	1	0	1	0.7	-0.1	-0.1	1	0	0.7	-0.1	-0.1
	1	1	1	0	0.7	-0.1	-0.1	0	0	0.7	-0.1	-0.1

Clasificadores basados en redes neuronales. Multi-Layer Perceptron (MLP)

- El aprendizaje de la estructura de una red neuronal requiere experiencia, aunque existen algunas guías
- Entradas: Por cada variable numérica o binaria/booleana se pone una neurona de entrada. Por cada variable nominal (con más de dos estados) se pone una neurona de entrada por cada estado posible.
- Salidas: Si es para predicción se pone una única neurona de salida por cada valor de la variable clase
- Capas ocultas. Hay que indicar cuántas capas y cuantas neuronas hay que poner en cada capa. Algunos comodines:
  - i/o: neuronas en la entrada/salida
  - t: i+o
  - a: t/2 (es el valor por defecto)
- Cuando la red neuronal está entrenada, para clasificar una instancia, se introducen los valores de la misma que corresponden a las variables de los nodos de entrada.
  - La salida de cada nodo de salida indica la probabilidad de que la instancia pertenezca a esa clase
  - La instancia se asigna a la clase con mayor probabilidad

Clasificadores basados en redes neuronales. Multi-Layer Perceptron (MLP)

- Ejemplo Iris:



Clasificadores basados en redes neuronales. Multi-Layer Perceptron (MLP)

- Todas las variables numéricas se normalizan [-1,1]

- Algoritmo de *Backprogation* o retropropagación:
  - Basado en la técnica del gradiente descendiente
  - No permite conexiones hacia atrás (retroalimentación) en la red

- Esquema del algoritmo de retropropagación
  1. Inicializar los pesos y sesgos aleatoriamente
  2. Para r=1 hasta número de *epoch* hacer
     a) Para cada ejemplo *e* de la BD hacer
     b) Lanzar un proceso *forward* de propagación en la red neuronal para obtener la salida asociada a *e* usando las expresiones vistas anteriormente
     c) Almacenar el valor O_j producido en cada neurona N_j
     d) Lanzar un proceso *backward* para recalcular los pesos y los sesgos asociados a cada neurona

1 *epoch* = procesamiento de todos los ejemplos de la BD

Clasificadores basados en redes neuronales. Multi-Layer Perceptron (MLP)

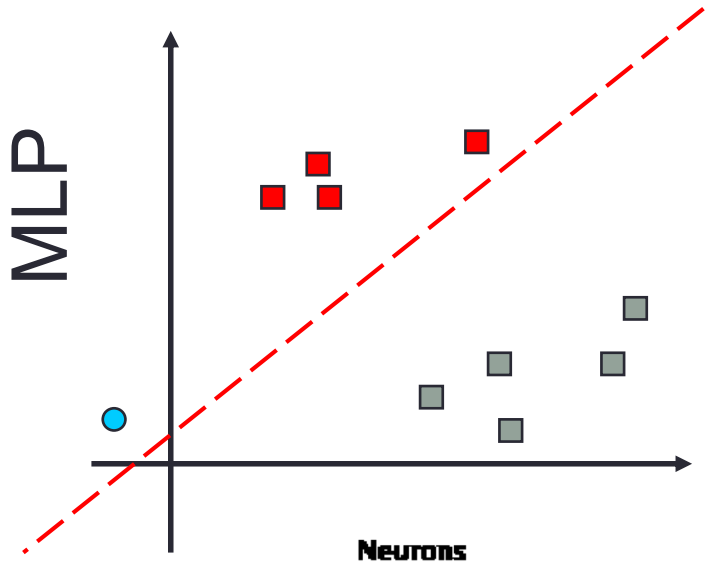
- Fase de retropagación:
  - Para cada neurona de salida N_s hacer BP(N_s)
  - BP(N_j):
    1. Si N_j es una neurona de entrada, finalizar
    2. Si N_j es una neurona de salida

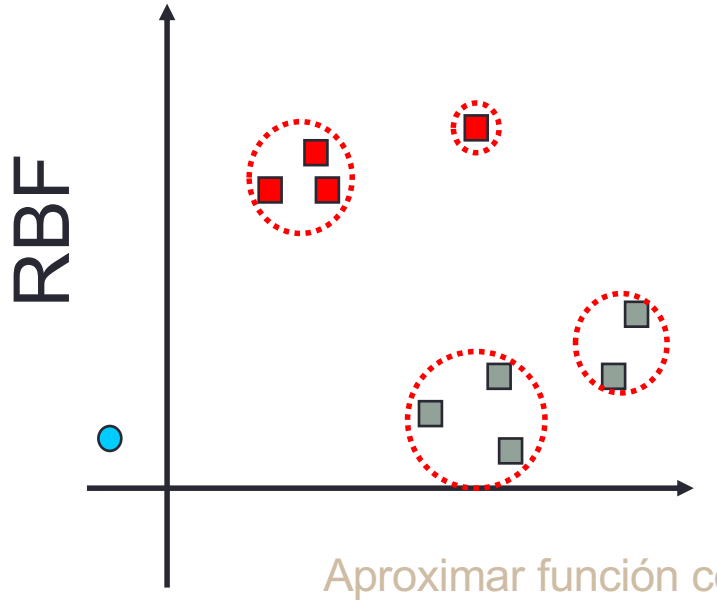
       entonces $Err_j = O_j \cdot (1-O_j) \cdot (T_j - O_j)$

       si no $Err_j = O_j(1-Oj) \Sigma_{k \text{ de salida}} Err_k \cdot w_{jk}$

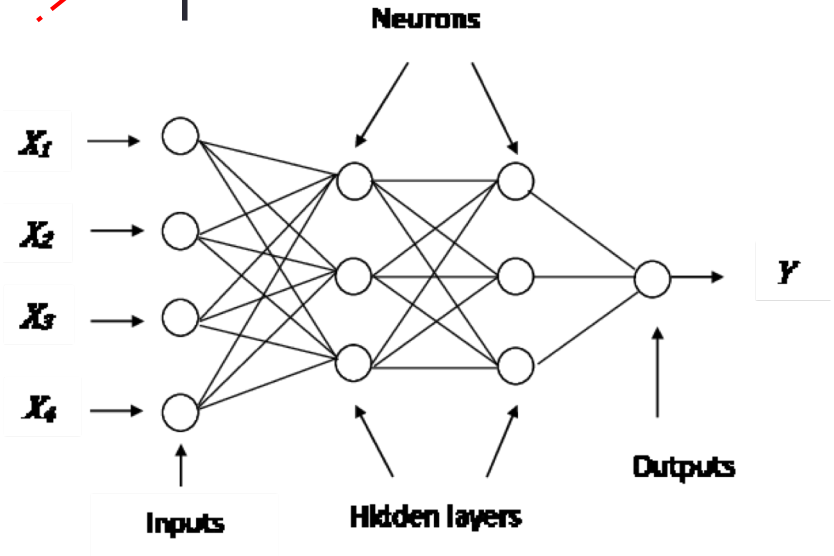
       con T_j el valor predicho en la neurona N_j
    3. Actualizar los pesos $w_{ij} = w_{ij} + \alpha \cdot Err_j \cdot O_i$
    4. Actualizar los sesgos: $\Theta_j = \Theta_j + \alpha \cdot Err_j$
    5. Para cada neurona N_i tal que $N_i \rightarrow N_j$ hacer BP(N_i)

Clasificadores basados en redes neuronales. Redes de Funciones de base Radial (RBF)





Aproximar función con combinación lineal de funciones de base radial



$$f(x) = \sum w_j h_j(x)$$

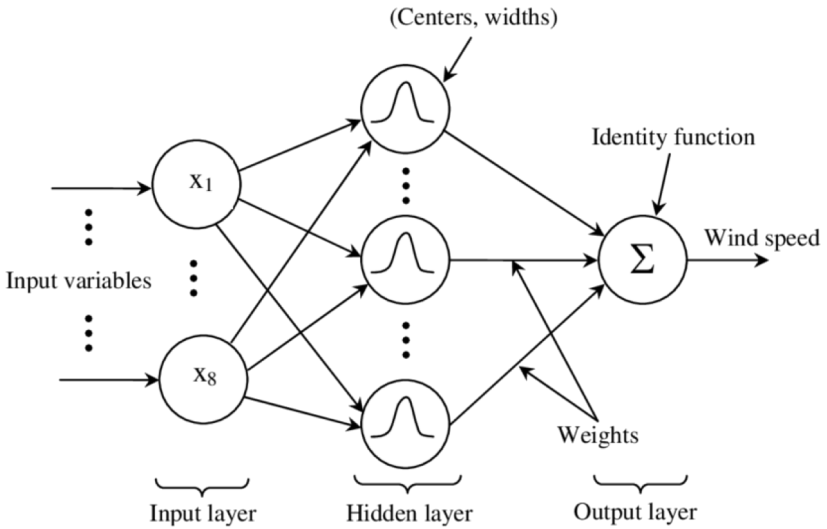
$$h_j(x) = \exp(-(x-c_j)^2 / r_j^2)$$

Where c_j is center of a region,

r_j is width of the receptive field

Clasificadores basados en redes neuronales. Redes de Funciones de base Radial (RBF)

- Diseño de una red neuronal como un problema de ajustado de curvas
- **Aprendizaje:** Encontrar la superficie en espacio multidimensional que mejor encaja con los datos de entrenamiento
- **Generalización:** Usar esta superficie multidimentional para interpolar los datos de test
- Tres Capas:
  - Capa de entrada
    - Nodos fuente que conectan la red a su entorno
  - Capa oculta
    - Unidades ocultas proporcionan un conjunto de funciones base
    - Alta dimensionalidad
  - Capa de salida
    - Combinación lineal de funciones ocultas



MLP Example with Keras: Iris

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
#
# Import Keras modules
#
from keras import models
from keras import layers
from keras.utils import to_categorical
#
# Create the network
#
network = models.Sequential()
network.add(layers.Dense(512, activation='relu',
input_shape=(4,)))
network.add(layers.Dense(3, activation='softmax'))
#
# Compile the network
#
network.compile(optimizer='rmsprop',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
#
# Load the iris dataset
#
iris = datasets.load_iris()
X = iris.data
y = iris.target
#
# Create training and test split
#
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, stratify=y, random_state=42)
#
# Create categorical labels
#
train_labels = to_categorical(y_train)
test_labels = to_categorical(y_test)
#
# Fit the neural network
#
network.fit(X_train, train_labels, epochs=20, batch_size=40)
```

RMSProp o *Root Mean Square Propagation* es una variación de AdaGrad en la que, en lugar de mantener un acumulado de los gradientes, se utiliza el concepto de "ventana" para considerar solo los gradientes más recientes.

https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/