MUBD

Màster Universitari en Enginyeria de Dades Massives (Big Data)

Estadística

Índice

Árboles condicionales (o de decisión)

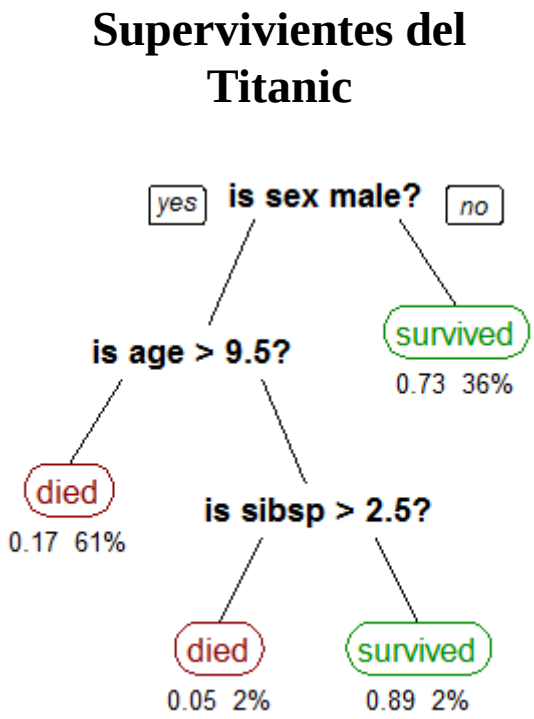
Introducción

- **Definición**: Tipo de algoritmo de partición recursiva. La respuesta puede ser una variable cuantitativa o cualitativa.
- **Fundamento**: A partir de un nodo raíz se van dividiendo los datos recursivamente en base a los valores de alguna variable que minimice algún criterio de optimalidad.
- **Tipos de nodos**:
  - **Raíz**. Nodo donde empieza un árbol
  - **Decisión**. Nodo donde se bifurca el árbol según los valores de alguna variable
  - **Hoja (o terminal)**. Nodo final del árbol donde se almacena alguna información resumen de los elementos que pertenecen a esa hoja
- Si la raíz es un nodo-hoja, entonces el árbol de decisión es **trivial o degenerado** y se hace la misma clasificación para todos los datos.

Árboles condicionales

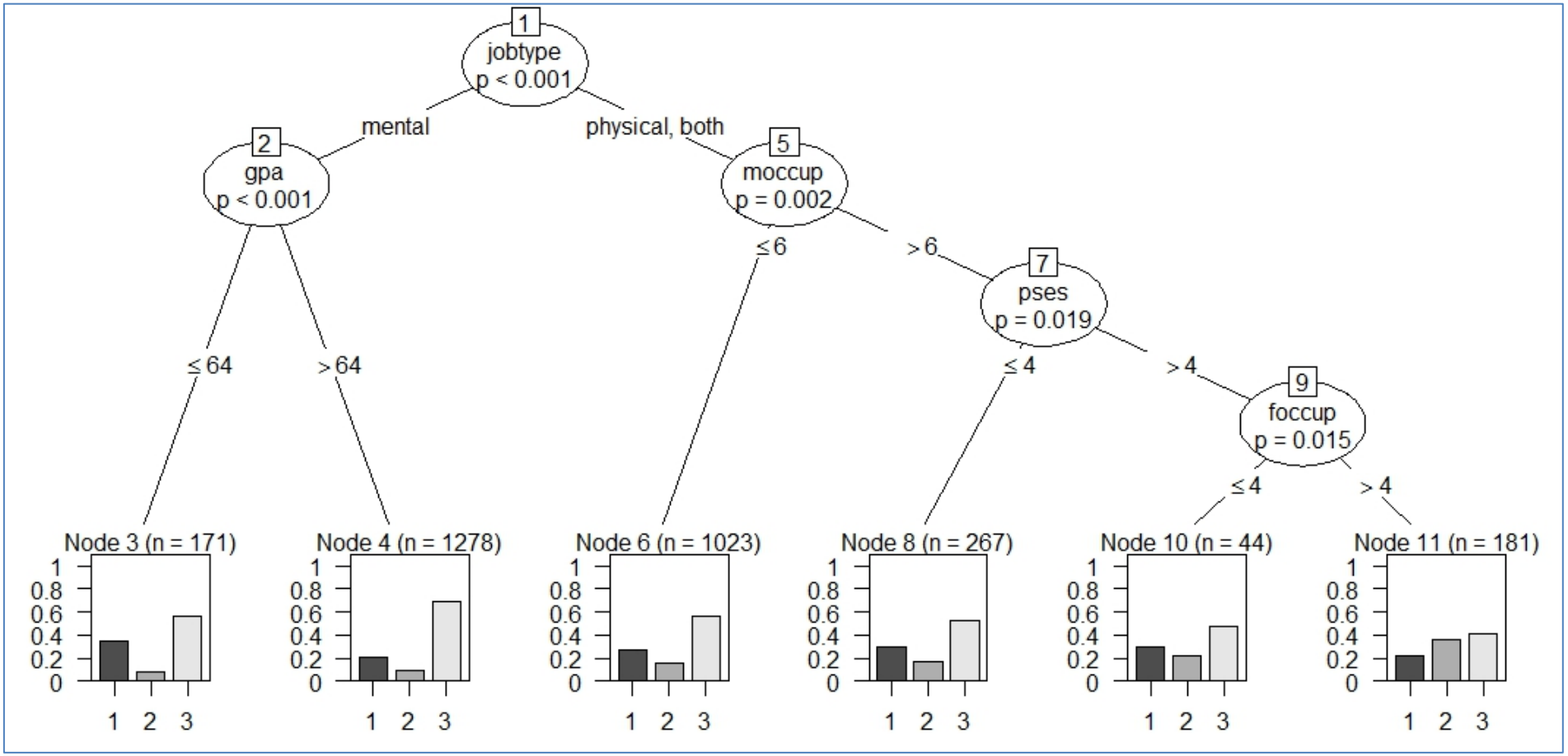
Introducción

- Para una determinada respuesta, se evalúa en cada nodo qué variable predictora discrimina mejor la respuesta
  - Para las variables cualitativas se realiza una separación de las categorías
  - Para las variables cuantitativas, se busca el punto de corte óptimo
- Cada nodo resultante se evalúa nuevamente con el mismo procedimiento



Árboles condicionales

Ejemplo R



Nota: Una variable puede aparecer más de una vez en el recorrido del nodo raíz al nodo hoja

Árboles condicionales

Algoritmo

- **Construcción del árbol**
  - Se empieza evaluando todas las variables de la muestra de entrenamiento.
  - Se selección aquella variable que optimiza algún criterio de partición previamente fijado (p.ej, el porcentaje de bien clasificados, el coeficiente gini, algún test estadístico….)
  - Se realiza esta partición y se vuelve a aplicar el procedimiento a los nodos-hijos
  - En los nodos hoja se almacena:
    - El valor mayoritario de los datos de entrenamiento si es una decisión de clasificación (o las probabilidades)
    - El valor medio de los resultados como una estimación de regresión
  - Se detiene la construcción cuando se cumple algún criterio de parada (ver siguiente diapositiva)
- **Predicción**
  - En cada nodo se evalúa la variable de decisión y nos movemos a otro nodo basado en el resultado de una comparación
  - Este proceso se repite hasta llegar a un nodo hoja.

Árboles condicionales

Criterios de parada

- **Basado en la discriminación del nodo**
  - P-valores ajustados: (p.ej., $p < 0.05$ ajustado por Bonferroni)
  - P-valores univariados (p.ej. $p < 0.05$)
  - Valores del estadístico
- **Fijados previamente por el usuario**
  - Profundidad máxima del árbol
  - Mínimo número de elementos para dividir un nodo
  - Mínimo número de elementos del nodo
  - …

Árboles condicionales

Pros y contras

Ventajas:

- Tanto las predictoras como la respuesta pueden ser continuas o categóricas
- Selección de variables automática
- Consideran las interacciones
- No tienen problemas con los datos ausentes
- Árboles pequeños fáciles de interpretar
- Poco sesgo

Inconvenientes:

- Capacidad predictiva mejorable
- Árboles grandes complicados de interpretar
- Mucha varianza

Árboles condicionales

R

- ***ctree*** {package: *partykit, party*}. Construcción del árbol.
- ***ctree_control***. Parámetro de la función *ctree* que a su vez permite listar los siguientes parámetros
  - ***teststat****: tipo de prueba*
  - ***testtype****: computación del test estadístico.*
  - ***mincriterion****: el valor de (1 – pvalor) a partir del cual se produce una partición*
  - ***minsplit****: mínimo número de elementos para considerar una partición*
  - ***minbucket****: mínimo número de elementos en un nodo hoja*
  - ***mtry****: número de variables a considerar en cada partición (0 = no seleccción)*
  - ***maxdepth****: máxima profundidad del árbol…*

Árboles condicionales

R – Otras funciones

El paquete ***party*** contiene otras funciones para trabajar con árboles de decisión.

- ***plot***: dibuja el árbol. Se puede fijar el formato del resumen de la respuesta en el nodo final. Ver *?plot.BinaryTree*
- ***predict***: realiza predicciones sobre un conjunto de datos dado un modelo de árbol. Puede predecir una categoría concreta; un valor concreto (*response*) o probabilidades sobre categorías (*probability*)

Árboles condicionales

Podado (prune)

- En ocasiones, podar el árbol puede evitar el sobre-ajuste
- La poda consiste en eliminar los nodos superfluos después de construir el árbol
- Funciones en R:
  - ***prune.misclass*** {package: tree}
  - ***nodeprune*** {package:partykit} lo realizan

Random Forest

Introducción

- **Definición**: consiste en la generación de muchos árboles condicionales a partir de versiones aleatorias del conjunto de entrenamiento.
- **Aleatoriedad de los árboles**. El objetivo es construir árboles que no estén correlacionados entre sí. Se consigue de dos formas:
  - **Bootstrap** de las observaciones para conseguir los conjuntos de entrenamiento
  - **Selección aleatoria de las variables** a considerar en cada división
- **Resultado**: Realiza un promedio de las probabilidades (respuesta categórica) o de las medias (respuesta continua) para cada una de las observaciones dentro de cada árbol.

Random Forest

Pros y contras

Ventajas:

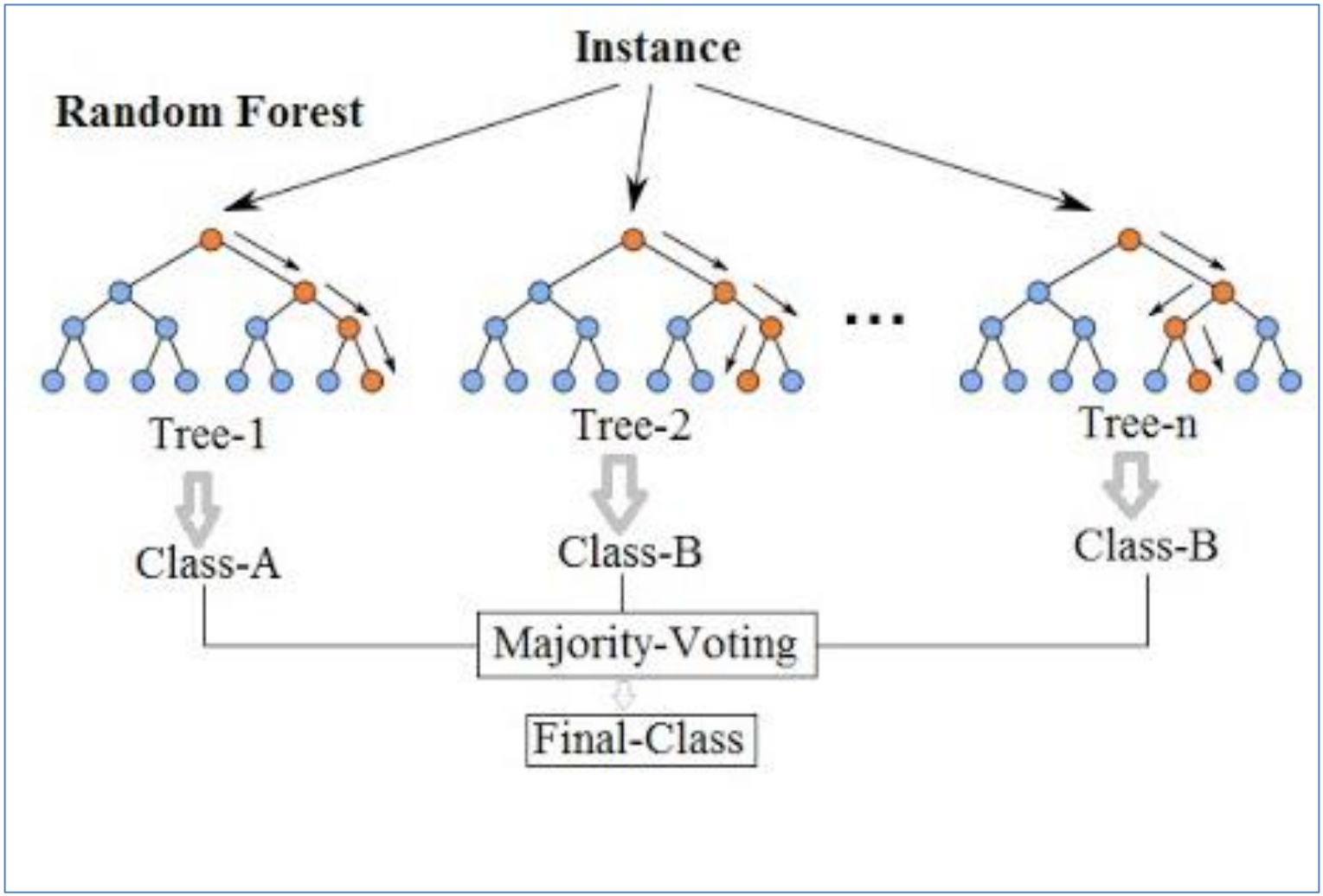
- Manejan variables continuas y categóricas indistintamente tanto predictoras como respuesta
- Selección de variables automática
- Consideran las interacciones
- No tienen problemas con los datos ausentes
- Gran capacidad predictiva
- Poco sesgo y poca varianza (disminuida a base del remuestreo)
- No requieren conjunto de entrenamiento.
- Cálculo de la importancia de las variables y del error de predicción "on the fly"

Inconvenientes:

- Relativa exigencia computacional
- Interpretación complicada

Random Forest

Ejemplo predicción



Random Forest

Algoritmo

1. Dado el conjunto de datos d = (*X*, *y*) donde X es el conjunto de variables predictoras e *y* la variable respuesta. Sea $m \leq p$ y el número de árboles B
2. Para cada árbol b = 1, 2,…, B hacer lo siguiente.
   a) **Crear una muestra de "*n*" observaciones** a partir de bootstrap (muestreo con remplazamiento de las "n" observaciones) que formará los datos de entrenamiento d_b^*.
   b) **Construir un árbol** de profundidad máxima $\hat{r}_b(x_i)$ usando los datos en d_b^* muestreando *m* variables al azar antes de hacer cada división.
   c) **Guardar el árbol**, así como las frecuencias de muestreo bootstrap para cada una de las observaciones en el conjunto de entrenamiento
3. Calcular la predicción para cualquier punto x_0 como un "promedio" de las predicciones para cada árbol donde no ha intervenido en la construcción.
4. Calcular el error de predicción (Out-Of-Bag) OOB_i para cada respuesta observada y_i. El error OOB global es el promedio de estos OOB_i.

Random Forest

Estimación del error de predicción (Out-of-bag)

- Para cada par (x_i, y_i), se promedian todas las predicciones hechas para x_i cuando no esté dentro de la muestra bootstrap:

$$\hat{r}_{\mathrm{rf}}^{(i)}(x_i) = \frac{1}{B_i} \sum_{b\,:\,w_{b\,i}^{*}=0} \hat{r}_b(x_i)$$

donde B_i es el número de veces que la observación *i* no está en la muestra bootstrap

- La tasa de error se computará como el promedio de las tasas de error para cada observación:

$$\mathrm{err}_{\mathrm{OOB}} = \frac{1}{n} \sum_{i=1}^{n} L[y_i, \hat{r}_{\mathrm{rf}}^{(i)}(x_i)]$$

donde *L* es la función de pérdida de interés, p.ej. el error de clasificación (respuesta cualitativa) o el error cuadrático medio (respuesta cuantitativa)

- El error de predicción depende de la correlación entre los árboles y de la capacidad predictiva individual de cada uno de ellos.

Random Forest

Estimación del error estándar

- Se usa el estimador de Jacknife

$$\widehat{\mathrm{V}}_{\mathrm{jack}}(\hat{r}_{\mathrm{rf}}(x_0)) = \frac{n-1}{n}\sum_{i=1}^{n}\left(\hat{r}_{\mathrm{rf}}^{(i)}(x_0) - \hat{r}_{\mathrm{rf}}(x_0)\right)^2$$

Donde $\hat{r}_{rf}^{(i)}(x_0)$ es el la estimación para x_0 usando todas las observaciones menos la *i-ésima* y $\hat{r}_{rf}(x_0)$ es el promedio de todas las estimaciones previas

- Esta estimación del error estándar de la estimación suele estar sesgada sobre todo para árboles pequeños. Existen versiones de la estimación de este error para aminorar este sesgo.

Random Forest

Cálculo de proximidades entre puntos

- Aparte de obtener predicciones, un resultado adicional factible que se puede obtener con los random forests son las similitudes entre individuos (a costa de un coste computacional extra)
- Después de cada árbol construido, se calcula la proximidad para cada pareja de observaciones
- Si una pareja ocupa el mismo nodo hoja (o terminal) su proximidad se incrementa.
- Al final, las proximidades se normalizan dividiendo por el número de árboles
- Las proximidades pueden ser útiles para la detección de *outliers*, para remplazar datos ausente, visualizar los datos en un dendograma,…

Random Forest

Importancia de las variables (interpretabilidad)

- El algoritmo random-forest es una "caja negra". Necesitamos alguna herramienta que nos ayude a interpretar los resultados.
- Se podría concluir que cualquier variable que nunca se utilice en los árboles es poco probable que sea importante, pero se debe tener un método para cuantificar la importancia relativa de las variables del conjunto de datos.
- Cada vez que se utiliza una variable en un árbol, el algoritmo almacena la disminución del error o el aumento de la desigualdad debido a esta partición.
- Estos valores de “disminución” o “aumento” se acumulan a lo largo de todos los árboles para cada variable, y servirán para sintetizar la importancia relativa.

Random Forest

Parámetros relevantes

Número de variables

- $m = \sqrt{p}$ *para respuesta categórica*
- $m = p/3$ *para respuesta continua*
- $m = 1$ para dar oportunidad a todas las variables a entrar
- El objetivo es construir árboles poco correlacionados entre sí (*m* pequeña) y con gran capacidad predictiva cada uno de ellos (*m* grande). Se debe encontrar un equilibrio.
- Generalmente el rango de optimalidad es bastante amplio

Número de árboles

- Cuántos más mejor. No se produce sobreajuste por usar más árboles.
- *ntree* = 500 es un valor razonable

Random Forest

R

- ***randomForest*** {package: *randomForest*}
  - ***ntree***: número de árboles
  - ***mtry***: número de variables a considerar en cada partición
  - ***nodesize***: tamaño mínimo de los nodos terminales
  - ***maxnodes***: número máximo de nodos terminales por árbol
  - ***importance***: ¿Se calcula la importancia de las variables?
- *tuneRF* {package: *randomForest*}. Cálculo del parámetro *mtry*
- ***cforest*** {package: *party*}
- ***CoreModel*** {package: *CORElearn*}

Random Forest

R – Otras funciones

El paquete ***randomForest*** contiene otras funciones interesantes.

- ***tuneRF***: encontrar el parámetro *mtry* óptimo
- ***varImpPlot***: importancia de las variables
- ***plot***: grafica los errores de predicción en función del número de árboles (ver ? plot.randomForest)

Random Forest

Consideraciones

- Si el **número de variables es muy grande** se realizan estos 2 pasos secuencialmente:
  - Realizar una única ejecución (pocos árboles) con todas las variables
  - Seleccionar únicamente las más importantes para realizar una ejecución más larga (más árboles)
- **Valores ausentes** en la muestra bootstrap. Existen 2 mecanismos de imputación
  - **Simple**. Imputar la mediana y la moda en las variables cuantitativas y cualitativas, respectivamente
  - **Compleja**. Se calculan las proximidades entre observaciones y se realiza la imputación ponderando por ellas (*rfImpute {randomForest}*).
- También sirven para hacer **clusterización no supervisada**
- En general, los **clústeres no balanceados** (*n* diferente) no afectan a la capacidad predictiva a no ser que exista una clase extremadamente no balanceada

Random Forest

Aplicaciones

- En general, para cualquier tipo de predicción
- Imputación de datos ausentes
- Clasificación de imágenes
- Tasa de conversión en página web
- Bajas de clientes

SVM – Support Vector Machine

Introducción

- **Definición**: Tipo de algoritmo de partición binaria no probabilístico.
- **Fundamento**: Dado un conjunto de entrenamiento del cuál se conozca las clases, se puede hallar un hiperplano que separe de forma óptima los puntos de cada clase.
- **Escenarios**:
  - Las 2 clases son linealmente separables en el espacio original. Se puede producir si el número de variables es muy superior al número de casos (p>>n)
  - Las 2 clases no son linealmente separables en el espacio original. Aquí hay 2 subopciones:
    - Dejar un cierto margen (margen blando o *soft margin*) para que ciertos puntos no queden bien clasificados por el hiperplano que los divide de forma óptima
    - Ampliar el espacio original mediante funciones Kernel para obtener una separación lineal óptima

SVM

Escenario linealmente separables

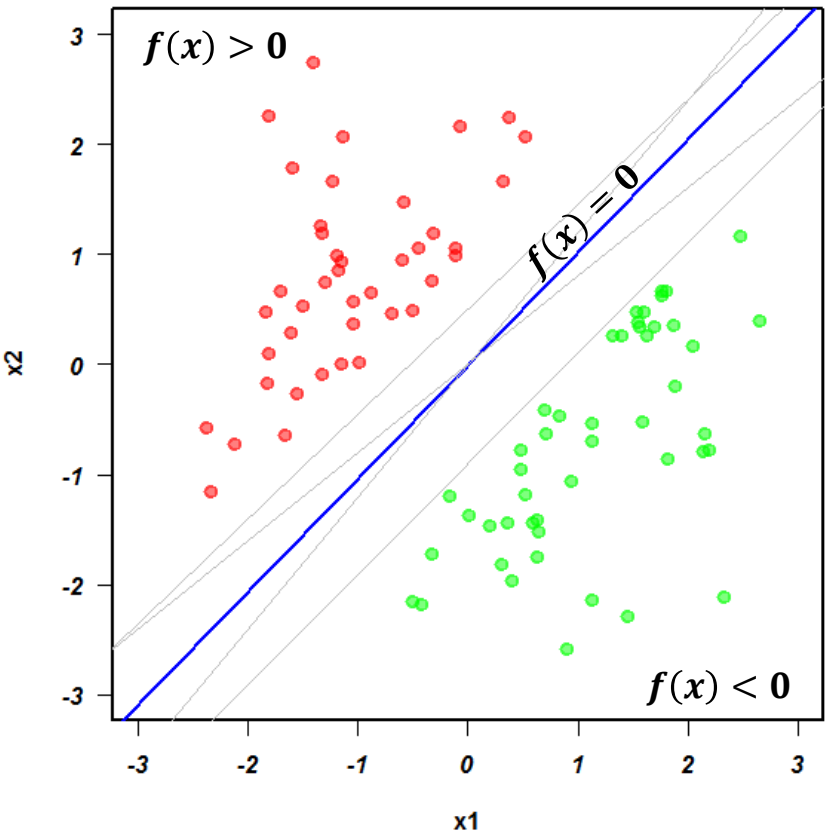
- Existen infinitos hiperplanos que pueden separar las clases correctamente
- La distancia de cualquier punto x_i al hiperplano es:

$$d_i = \frac{1}{\|\beta\|_2} \cdot y_i \cdot f(x_i)$$

β: coeficientes del hiperplano
y_i: clase de la observación *i (-1 o 1)*
f(x_i): Clasificación del punto en función del hiperplano:

$$f(x_i) = \beta_0 + \beta' \cdot x_i$$

- La regresión logística (alternativa a los SVM) falla en estimar los coeficientes en estos escenarios ya que el algoritmo de maximización de la verosimilitud no converge.



SVM

Escenario linealmente separables

Nota:
$f(x_i) = \beta_0 + x_i \cdot \beta_1 \geq 0 \quad si \quad y_i = +1$
$f(x_i) = \beta_0 + x_i \cdot \beta_1 \leq 0 \quad si \quad y_i = -1$
$y_i \cdot f(x_i) = y_i \cdot (\beta_0 + x_i \cdot \beta_1) \geq 1 \quad siempre$

- Se debe maximizar el margen (M) o la distancia mínima a cualquiera de los puntos
- Es decir, se ha de resolver el siguiente problema de optimización:

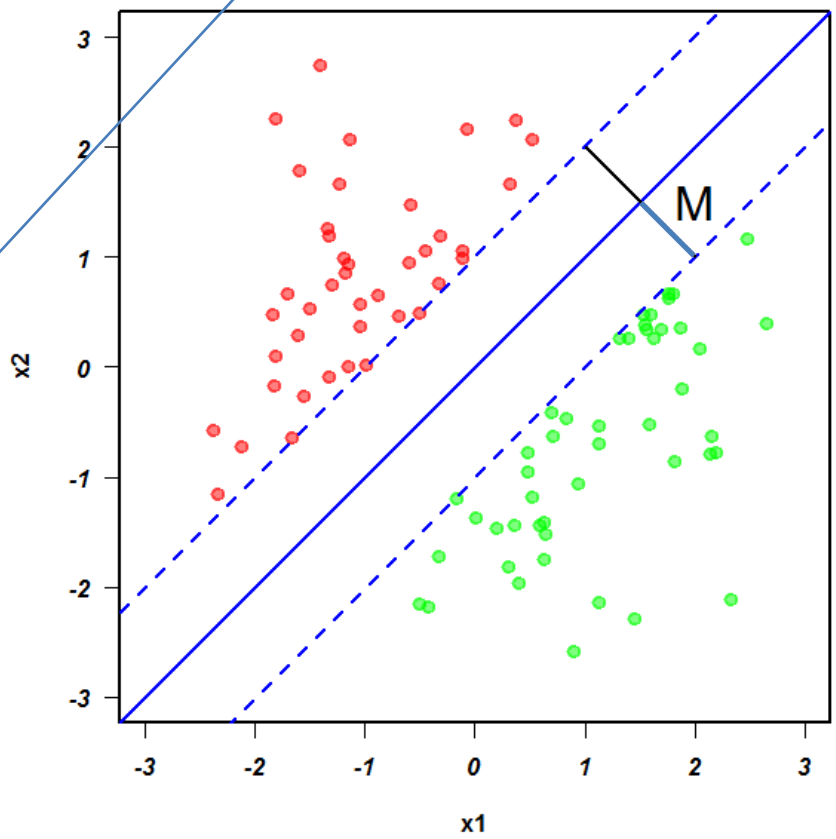
$$\underset{\beta_0,\beta}{\text{maximizar}} \quad M$$

$$sujeto\ a \quad \frac{1}{\|\beta\|_2} \cdot y_i \cdot f(x_i) \geq M \quad i = 1, \ldots, n$$

- Que es equivalente a:

$$\underset{\beta_0,\beta}{\text{minimizar}} \quad \|\beta\|_2$$

$$sujeto\ a \quad y_i \cdot f(x_i) \geq 1 \quad i = 1, \ldots, n$$



SVM

Escenario linealmente separables

- Es un problema de optimización cuadrática con restricciones lineales que no presenta dificultades
- La solución del problema de optimización es:

$$\hat{\beta} = \sum_{i \in S} \hat{\alpha}_i \cdot x_i$$

- Donde S es el conjunto de Soporte formado por el conjunto de puntos que están exactamente en el margen (como mínimo habrá uno)
- La implicación directa de esto es que la solución únicamente depende de estos puntos y el resto se puede mover libremente por la región correspondiente a su clase sin afectar al resultado.

SVM

Márgenes blandos

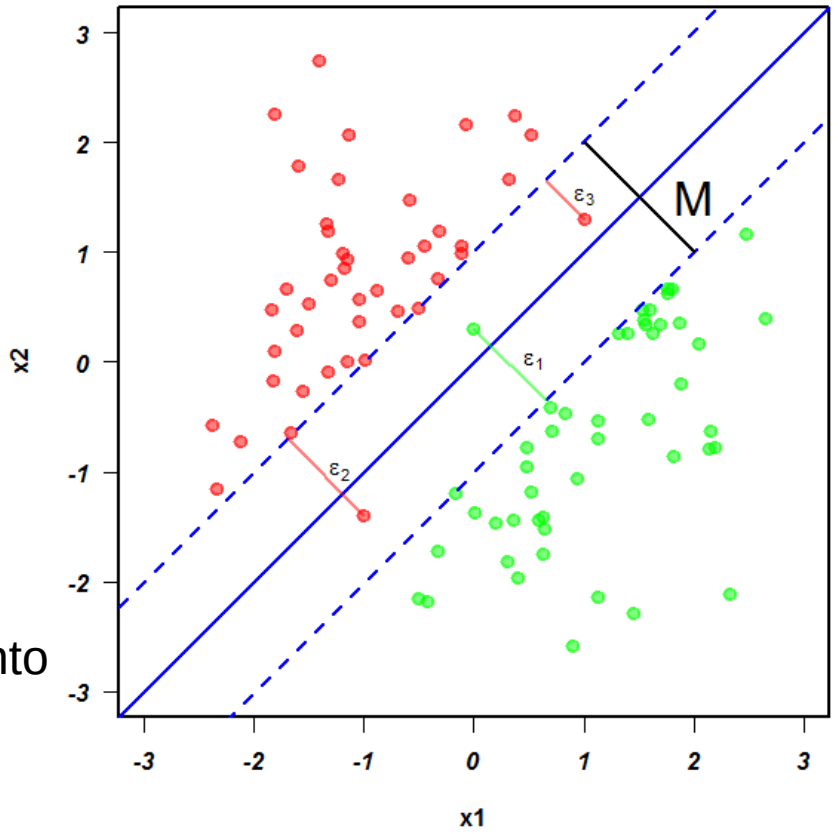
- Los datos que dispongamos pueden no ser linealmente separables
- La generalización del escenario anterior es aquella en que se permite a ciertos puntos violar el margen

$$\underset{\beta_0,\beta}{\text{minimizar}} \quad \|\beta\|_2$$

$$sujeto\ a \quad y_i \cdot f(x_i) \geq 1 - \epsilon_i \quad i = 1, \ldots, n$$

$$\epsilon_i \geq 0 \quad i = 1, \ldots, n$$

$$\sum_{i=1}^{n} \epsilon_i \leq B$$



- Donde B es la cantidad total de solapamiento permitida.

SVM

Márgenes blandos – Parámetro B

- En este caso, la solución del problema de optimización vendrá dada tanto por los puntos en el margen como por aquellos que lo violen (estos puntos constituirán el soporte)
- Cuanto mayor sea B, un mayor número de puntos tendrán influencia en la solución
- Por tanto, cuanto mayor sea el parámetro B se tendrá menor varianza
- Incluso en espacio linealmente separables puede introducirse este parámetro para obtener resultados más estables.
- Es un parámetro de control del sobreajuste

SVM

Criterio de función de pérdida penalizada

- El problema de optimización se puede formular en una única función de pérdida sin restricciones pero añadiendo una penalización

$$\min_{\beta_0,\beta} \sum_{i=1}^{n} [1 - y_i(\beta_0 + \beta \cdot x_i')]_+ + \lambda \cdot \|\beta\|_2^2$$

Parte nueva respecto a los márgenes rígidos

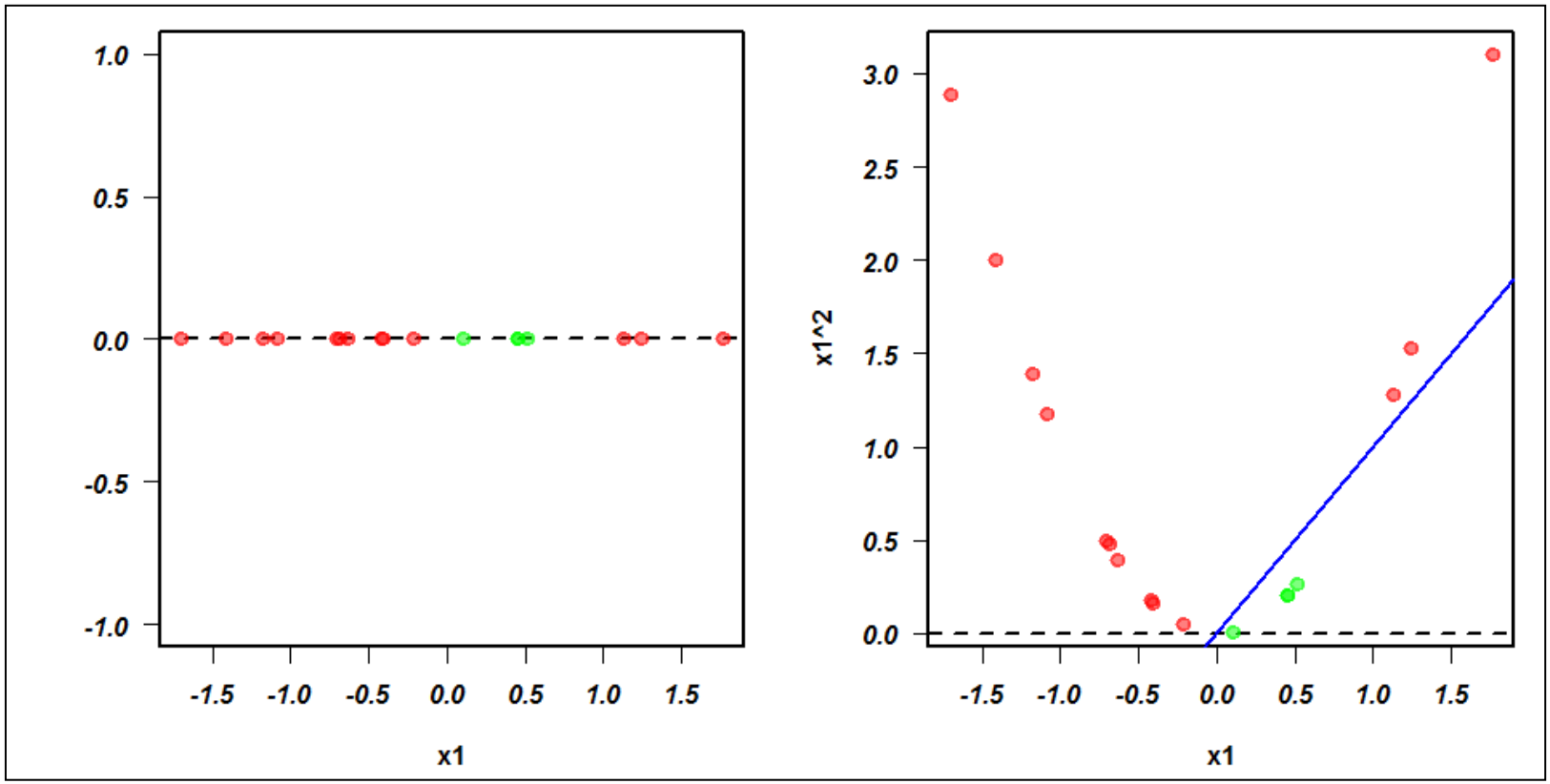
- La primera parte de la expresión:
  - Es una función lineal a trozos
  - Representa el coste de estar en el lado incorrecto del margen
  - Vale 0 para los puntos correctamente clasificados
- La segunda parte de la expresión:
  - Un λ grande se identifica con un parámetro B grande
  - Para datos separables, la solución óptima se obtiene para el caso $\lambda=0$

SVM

Mapeo en dimensiones superiores

- La idea intuitiva que hay detrás es que se puede aumentar la dimensión del espacio original para lograr tener un conjunto linealmente separable.

$$Sea\ \ \Phi: \mathcal{R} \rightarrow \mathcal{R}^n \quad tal\ que \quad \Phi(x) = (x, x^2)$$



SVM

Funciones Kernel (I)

- Siempre es posible mapear los datos en un espacio de dimensión superior donde sean linealmente separables.
- La ecuación del hiperplano, sin embargo, se basa en el producto escalar de dos vectores (puntos):

$$\hat{f}(x) = \beta_0 + x'\hat{\beta} = \beta_0 + x' \sum_{i \epsilon S} (\hat{\alpha}_i \cdot x_i) = \beta_0 + x' \sum_{i \epsilon S} (\hat{\alpha}_i \cdot \langle x, x_i \rangle)$$

- El coste computacional de hallar la solución SVM depende **linealmente del número de variables** pero hasta de forma **cúbica del número de observaciones**.
- ¿Necesitamos mapear todos los puntos? NO, solo los productos escalares. Una **función Kernel** es alguna función que se corresponde con un producto escalar en el espacio ampliado.

SVM

Funciones Kernel (II)

- Las funciones Kernel logran que el coste de calcular estos productos escalares sea similar a la del espacio original.
- Las funciones más usuales son:
  - Polinómica:

$$K_d(x, z) = (1 + \langle x, z \rangle)^d$$

  - Radial:

$$K(x, z) = e^{-\gamma \|x - z\|_2^2}$$

  - Sigmoide:

$$K(x, z) = \tanh(\beta_0 x' z + \beta_1)$$

- Estas funciones Kernel pueden substituir el producto escalar en forma matricial:

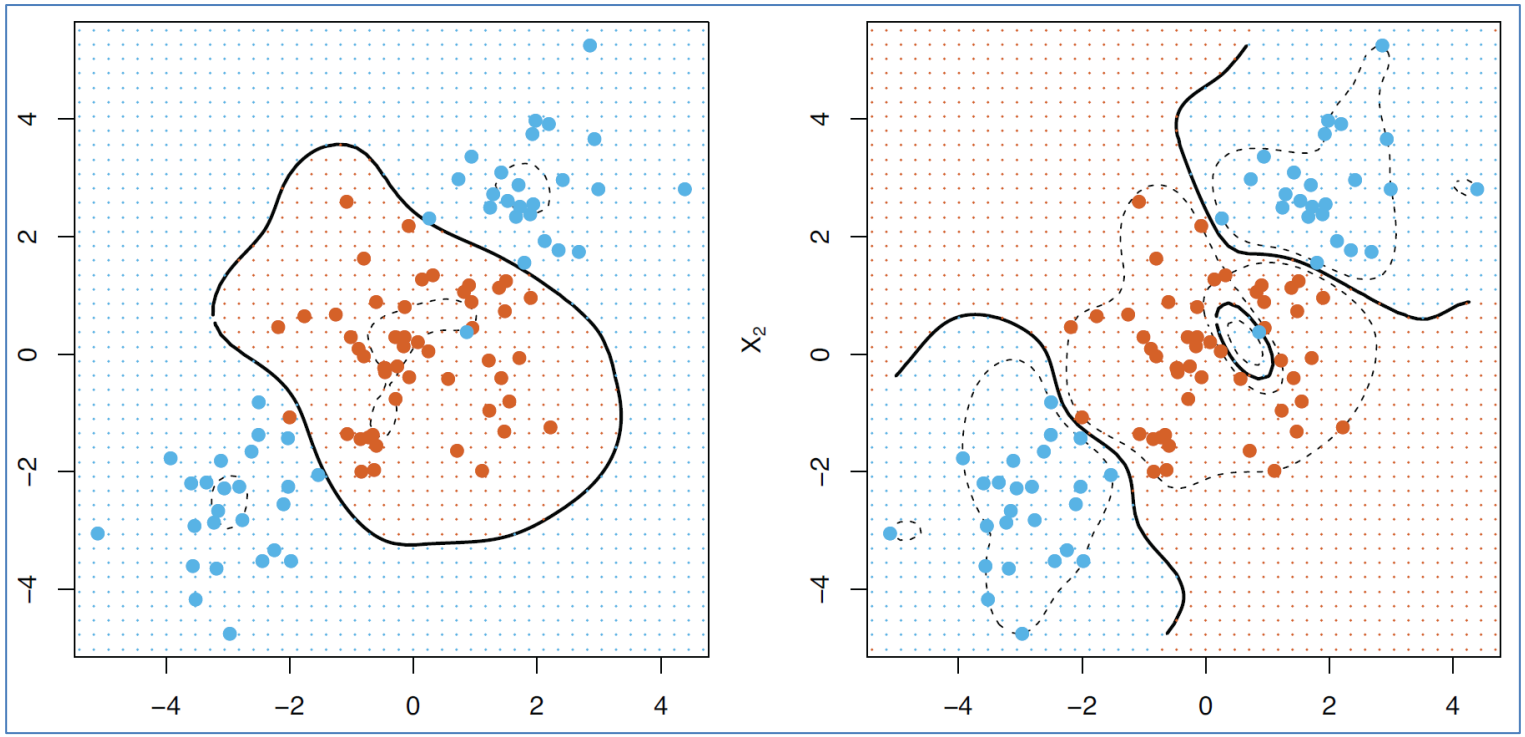
$$\hat{f}(x) = \beta_0 + x' \sum_{i \epsilon S} (\hat{\alpha}_i \cdot K(x, x_i))$$

- Toda función que de lugar a una matriz semi-definida positiva es una función Kernel

SVM

Funciones Kernel (III) - Ejemplo

- Estas dos figuras representan los límites de decisión (línea sólida) y los márgenes (línea punteada)
- El gráfico de la izquierda tiene una λ (~B) mayor y el de la derecha menor (sobreajuste)



SVM

Pros

- Flexibilidad en escoger una función kernel de similitud
- Solo unos pocos puntos (soporte) se usan para hallar la solución
- Capacidad para trabajar con espacios de gran dimensionalidad
- Tiene un parámetro para controlar el sobreajuste (B)
- Es un problema de optimización que siempre converge
- Es útil en situaciones donde tengamos muchas variables (incluso más que datos)

SVM

Contras

- Es terriblemente sensible al ruido. Unos cuantos ejemplos mal clasificados pueden hacer decrecer la capacidad predictiva.
- La capacidad predictiva es muy dependiente de los parámetros (probar varios escenarios)
- En principio, la formulación original sólo sirve para 2 clases. Alternativas para K>2 clases:
  - Construir un clasificador para cada una de las $\binom{K}{2}$ combinaciones. En la predicción de una nueva observación, asignar la clase donde haya sido asignado más veces. *svm* {package: *e1071*} implementa esta estrategia
  - Construir K clasificadores de cada clase contra el resto. Asignar una nueva observación a la clase que el punto este más alejado del hiperplano.
- No proporciona probabilidades. Es (en principio) únicamente determinista. Pueden hallarse probabilidades basadas en distancias.
- Interpretación complicada
- Para n's grandes tarda demasiado en converger

SVM

R

- ***svm*** *{package: e1071}. Support vector machine (plus kernel)*
- ***ksvm*** {package: *kernlab*}. Kernel – SVM
- Otras funciones:
  - *tune* {*package: e1071}*

SVM

Aplicaciones

- Categorizaciones de textos
- Clasificación de imágenes
- Bioinformática (clasificación de proteínas)
- Genética
- Reconocimiento de caracteres

Comparación de algoritmos (con comentarios)

	KNN	Naive Bayes	Árboles Condicionales	Random Forest	Observaciones
Tiempo computacional	✗	✓	✓	✗	KNN es peor que RF para conjuntos grandes de datos
Número de variables	✗	✓	✓	✓	KNN limitado a 20 variables
Independencia	✓	✗	✓	✓	NB tiene la premisa de independencia de los factores
Interacciones	✓	✗	✓	✓	NB no considera interacciones de las predictoras con la respuesta
Sesgo	✓	✗	✓	✓	NB tiende a estar sesgado en la predicción de las probabilidades
Varianza	✗	✓	✗	✓	RF es el único que maneja bien sesgo y varianza
Error medio	✗	✓	✓	✓	Mezcla de sesgo y varianza
Regiones de decisión	✓	✗	✓	✓	NB tiene problemas si los límites no tienen ciertas formas
Atributos irrelevantes	✗	✓	✓	✓	KNN considera todos los atributos por igual a no ser que se ponderen
Interpretabilidad	✗	✓	✓	✗	En general, los 4 son poco interpretables
Parámetros	✗	✓	✗	✗	Excepto NB, el resto exigen parámetros de entrada para ejecutarse
Datos ausentes	✗	✓	✓	✓	KNN es el único que no sabe operar con datos ausentes

Comparación de algoritmos

	KNN	Naive Bayes	Árboles Condicionales	Random Forest	SVM
Tiempo computacional	✗	✓✓	✓	✗	✗
Número de variables	✗	✓	✓	✓✓	✓✓
Independencia	✓	✗	✓	✓	✓
Interacciones	✓	✗	✓✓	✓✓	✓
Sesgo	✓	✗	✓	✓✓	✓
Varianza	✗	✓	✗	✓✓	✗
Error medio	✗	✓	✓	✓✓	✓✓
Regiones de decisión	✓✓	✗	✓	✓	✓✓
Atributos irrelevantes	✗	✓	✓	✓	✗
Interpretabilidad	✗	✓	✓	✗	✗
Parámetros	✗	✓	✗	✗	✗
Datos ausentes	✗	✓	✓	✓	✗

MUBD

Màster Universitari en Enginyeria de Dades Massives (Big Data)

Estadística